VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

STROJOVÉ UČENÍ VE STRATEGICKÝCH HRÁCH

MACHINE LEARNING IN STRATEGIC GAMES

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE — **MICHAEL VLČEK**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Strojové učení v současnosti diktuje pokrok umělé inteligence v soupeření s člověkem v rámci strategických her, ať už jde o šachy, Go, či poker. Oblastí strojového učení, která vykazuje nejperspektivnější výsledky ve hraní strategických her, je posilované učení. Velkým milníkem se pro současný vývoj stává počítačová hra Starcraft II, která svou komplexností mnohonásobně předčí dosavadní úspěchy v tomto oboru. Tato práce se zabývá rozborem problematiky, a navrhuje řešení prostřednictvím algoritmu posilovaného učení A2C a implementace optimalizace hyperparametrů PBT (trénování na bázi populace), které může být pro dosavadní výsledky krokem vpřed.

Abstract

Machine learning is spearheading progress for the field of artificial intelligence in terms of providing competition in strategy games to a human opponent, be it in a game of chess, Go or poker. A field of machine learning, which shows the most promising results in playing strategy games, is reinforcement learning. The next milestone for the current research lies in a computer game Starcraft II, which outgrows the previous ones in terms of complexity, and represents a potential new breakthrough in this field. The paper focuses on analysis of the problem, and suggests a solution incorporating a reinforcement learning algorithm A2C and hyperparameter optimization implementation PBT, which could mean a step forward for the current progress.

Klíčová slova

Strojové učení, posilované učení, Starcraft II, částečně pozorovatelný Markovův rozhodovací proces, neuronová sít, SC2LE, A2C, A3C, trénování na bázi populace, optimalizace hyperparametrů, strategie, agent.

Keywords

Machine learning, reinforcement learning, Starcraft II, partially observable Markov decision process, neural network, SC2LE, A2C, A3C, Population Based Training, hyperparameter optimization, strategy, agent.

Citace

VLČEK, Michael. *Strojové učení ve strategických hrách.* Brno, 2018. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Strojové učení ve strategických hrách

Prohlášení

Prohlašuji, že jsem tuto závěrečnou práci vypracoval samostatně pod vedením pana Doc. RNDr. Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michael Vlček
22. května 2018

Poděkování

Chtěl bych velmi poděkovat svému vedoucímu, panu Doc. RNDr. Smrži, Ph.D, za jeho odbornou pomoc a trpělivost s ohledem na můj zahraniční studijní pobyt po celý zimní semestr.

Obsah

Kapitola 1

Úvod

Strojové učení (SU) je v dnešní době stále žhavým tématem nejen v odborných kruzích, ale je i jedním z témat, která se dostanou do rukou i širšímu publiku, a to hlavně díky velmi aktivnímu výzkumu a širokému spektru možnosti využití této technologie v praxi. Ať už jde například o rozsáhlé optimalizační systémy pro rozpoznávání obrázků, analýzu přirozené mluvy a textu, nebo vysledování vzorů chování zákazníku z velkého množství dat, strojové učení nabízí obrovský potenciál, a to nejen v komerční sféře. Právě proto přetrvává snaha o posouvání hranic možností algoritmů SU, přestože se často jedná o velmi časově i výpočetně náročné projekty. Jedním s příkladů snahy o posunutí těchto hranic je soupeření algoritmů s lidským protějškem v různých strategických hrách.

Tato práce pojednává nejdříve krátce o umělé inteligenci jako takové, což nám zasadí do kontextu pozdější užší zaměření na strojové učení a jeho vztah se strategickými hrami, protože právě soubojem o prvenství mezi strojem a člověkem (zejména z pohledu algoritmizace hraní her prostřednictvím SU) se práce zabývá hned poté. V kapitole 3 se pak seznámíme blíže s teoretickými základy potřebnými k vysvětlení principů vlastního řešení, načež se kapitola 4 soustředí na počítačovou strategickou hru Starcraft II (dále jen SC2), která bude předmětem pro zkoumání použitelnosti vlastního algoritmu. Podíváme se na jednotlivé aspekty hry SC2, popis prostředí a pravidel, podmínek vítězství, apod. Hned poté je kapitolou 5 přiblížen postup navázání rozhraní mezi algoritmem ("hráčem") a herní smyčkou tak, aby byla tato umělá inteligence schopna tuto hru bez potíží ovládat. V kapitole 6 pak navážeme přehledem dosavadních úspěchů SU v této hře, včetně krátkého popisu jednotlivých algoritmů, které se o tyto úspěchy dělí. V další části už se vyskytuje popis návrhu a implementace zmiňovaného vlastního řešení (kapitola 7), po čemž už zbývá jen pustit se do experimentů v kapitole 8. Nakonec je vyvozen závěr, shrnující poznatky z práce a doporučení dalšího postupu.

Kapitola 2

Umělá inteligence

2.1 Zrod UI

Přestože první myšlenky týkající se umělé inteligence mají kořeny již v 14. století, obor umělé inteligence jako takový byl založen až v roce 1956 na Dartmouth College v New Hampshire. John McCarthy, který je ve světě považován za zakladatele UI, ustanovil tento pojem jako vědu zabývající se vytvářením inteligentních strojů. Se zrodem UI je však spojen i John von Neumann. Na jím navrhnutém počítači *JOHNNIAC* poprvé běžel program sestavený Allenem Newellem, Cliffem Shawem a Herbertem Simonem, známý jako *Logic Theorist*. Ve stejném roce ještě před vytvořením oboru tedy spatřil světlo světa stroj schopný vyhodnocení primitivních logických teorémů [11]. Byl schopen dokázat 38 z prvních 52 teorému v knize o klíčových principech a základních stavebních kamenech matematiky *Principia Mathematica*, a v některých případech nalézt nové, elegantnější důkazy pro tyto teorémy [25].

2.2 UI v současnosti

Zájem o umělou inteligenci stále roste, a v dnešní době lze její uplatnění nalézt v mnoha oborech lidské působnosti. Tato technologie se těší značné pozornosti v oblastech jako je např. letectví, finance, medicína, vzdělávání, HR, marketing, doprava, telekomunikace, a dokonce i umění.

Pokud se zaměříme například na medicínu, jen ve fázi diagnózy lze nalézt využití pro umělé neuronové sítě při klinické diagnóze, analýze obrazu v radiologii a histopatologii, interpretaci dat v prostředí oddělení intenzivní péče a analýze signálu [35]. Umělá inteligence díky své schopnosti vyvozovat smysluplné vztahy v obsáhlých databázích umí diagnostikovat, navrhovat léčbu a předpovídat průběh mnohých onemocnění. Studie provedená Williamem G. Baxtem založená na retrospektivní diagnóze akutního infarktu myokardu u 356 pacientů je často citovaným příkladem využití UI v tomto oboru. Ze skupiny 236 pacientů, kteří neměli infarkt myokardu, a skupiny 120 pacientů, kteří jej měli, byla síť natrénována na datech o polovině pacientů z obou skupin. Ze zbývajících pacientů pak byla síť schopna identifikovat s 92% přesností pacienty s infarktem a 96% pacientů, kteří infarkt neměli. I po vyřazení pacientů s existujícím elektrokardiografickým záznamem byla síť úspěšná v 80% případů, což se ukázalo jako nesrovnatelně lepší než jakýkoliv dříve používaný postup. [5]

Tím ovšem funkce UI v medicíně zdaleka nekončí. Do UI spadá i oblast robotiky, a robotické systémy mohou usnadnit práci například při radiochirurgii (detekce a léčba zhoub-

ných i benigních nádorů prostřednictvím precizního ozařování), chirurgii obecně (asistence při provádění náročných zákroků vyžadující například extrémní přesnost), případně při jiných úkonech (navrhování přesného dávkování léčiv, odstraňování lidských chyb způsobených vyčerpáním/stresem, zrychlení procesu diagnózy, aj.). Díky své robustnosti jsou postupy (jako zmiňované umělé neuronové sítě) často přenositelné do mnoha odvětví a problémových domén, což je velmi důležitá vlastnost umělé inteligence, a představuje jednu z fundamentálních vlastností konceptu tzv. *Artificial General Intelligence*, což lze doslovně přeložit jako obecná umělá inteligence.

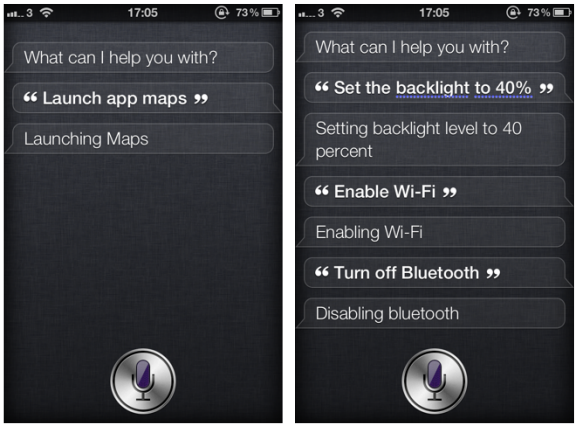
2.3 Obecná umělá inteligence

Všechny úspěchy v tomto odvětví si kladou za cíl přibližovat UI svými schopnostmi vnímání a konání čím dál více skutečnému životu, a na časové linii je tento záměr vidět v podobě postupného navyšování komplexity řešených problémů. Vzhledem k obecné povaze problémů hraní strategických her je pak možné tyto technologie přenést do jiných, výše zmíněných odvětví. Tyto důvody přímo souvisí se snahou o dosažení obecné umělé inteligence (dále ji ve zkratce budeme označovat jako AGI). Jedná se o UI umožňující přenesení své schopnosti řešit problémy v jedné doméně na problémy jiné domény. To by znamenalo rapidní nárůst její znovupoužitelnosti, zmenšení velikosti dat nutných k trénování, a umožní takovému modelu problémy zobecňovat. Z praktického hlediska by byla UI schopna využívat konceptu, kterému v češtině říkáme selský rozum. Každý jednotlivý úspěch v tomto odvětví znamená krok kupředu směrem k AGI.

Myšlenky nad vytvořením či existencí umělé inteligence schopné racionálního uvažování na úrovni průměrného člověka sahají až do starověké mytologie. Pouhých několik let po zrodu UI v 50. letech minulého století se i někteří tehdejší přední experti nechali slyšet, že nejpozději v průběhu jedné generace bude problém vytvoření AGI z velké části vyřešen [12, 13]. To se však, jak již dnes víme, nestalo. Od svých velmi optimistických výhledů do budoucnosti střídající se s opakovanou ztrátu zájmu o daný obor a snížení investic do výzkumu (období tzv. "AI winter") byl potenciál tohoto oboru neustále střídavě zpochybňován a vyzdvihován [22].

Absence skutečně smýšlejícího stroje však nebránila v uvažování, jak by bylo možné vyhodnocovat, zda se opravdu jedná o obecnou umělou inteligenci. Ve skutečnosti byl Alanem Turingem navržen tzv. *Turingův test* již v roce 1950. V zásadě se jedná o test zaměřený na věrohodnou simulaci konverzace s lidským protějškem v čistě textově založeném prostředí. Test se opíral o poměrně odvážnou hypotézu, protože zjevně není nutné, aby AGI byla schopna přesné simulace lidské inteligence [16] (nedává smysl předpokládat, že by stroj bez vlastního lidského těla měl konverzovat o stárnutí, hladovění nebo jiných tématech týkajících se lidského těla na úrovni člověka).

Patnáct let po navržení této imitační hry uměl program *Eliza* německo-amerického profesora Josepha Weizenbauma vést konverzaci s člověkem, přičemž využíval několika triků, z nichž nejdůležitější roli hrálo odpovídání na otázky prostřednictvím otázek. Způsob, jakým Eliza komunikovala se svým konverzačním partnerem, připomínal psychoterapeutický rozhovor. Eliza vykazovala chování bez jakékoliv známky po osobnostních znacích, a tedy se konverzace s ní do značné míry podobá chování rogerianského terapeuta. Další program, *Parry*, simuloval paranoidní chování prostřednictvím sledování vlastních emocí v podobě stavů napříč několika různými dimenzemi. Při porovnání transkripce rozhovorů mezi doktory a pacienty trpící paranoiou nebo Parrym nebyli psychiatři ani inženýři schopni rozlišit pacienta trpícím paranoiou a Parryho. [24] Další kroky ve vývoji se zaměřovaly buď na další



Obrázek 2.1: Siri (stejně jako někteří ostatní hlasoví asistenti) přijímá hlasové příkazy v přirozené řeči, a v dnešní době již dokáže rozeznat i poměrně složité příkazy a provádět je. Převzato z [30].

pokrok v úspěšnosti simulace, nebo přechod z textové komunikace na hlasovou. V devadesátých letech výzkum v oblasti zpracování řeči a přirozené komunikace značně zrychlil, a světlo světa spatřili další chatboti (*Thoughts*, *Dr. Sbaitso*, *Jabberwacky*, *A.L.I.C.E.*, aj.). Do chatbotů vstupovaly metody strojového učení, heuristiky pro pattern matching, fuzzy logika, atd. Velkým milníkem se stal rok 2011, kdy *Watson* od společnosti IBM dokázal zvítězit nad dvěma předešlými šampiony ve slavné americké televizní soutěži Jeopardy! s pomocí zpracování přirozené řeči a strojového učení nad obrovským množstvím dat. Na této frontě v současnosti vedou hlasoví asistenti v mobilních telefonech nebo v jiných zařízeních (*Siri*, *Google Assistant*, *Cortana*, *Alexa*). Facebook na své Messenger platformě dokonce umožňuje vývojářům vytvářet boty, kteří pak komunikují s uživateli. Nechvalným příkladem se však v roce 2016 stal chatbot *Tay* od společnosti Microsoft, který byl vypuštěn na sociální síti Twitter za účelem experimentování s učením bota na síti prostřednictvím interakce s ostatními uživateli, který musel být za méně než 24 hodin vypnut kvůli svým velmi kontroverzním zprávám.

Turingův test stále plní důležitou roli při hledání AGI, ale jeho správnost je zpochybňována i vyzdvihována dodnes [37, 28, 15]. Kritici Turingovi mimo jiné často vyčítají, že jeho test nemůže být ukazatelem skutečně obecné inteligence, jelikož definice testu v žádném ohledu negarantuje rozpoznání obecně inteligence jako takové, nýbrž kulturně závislé lidské inteligence. Tyto teorie se opírají o neoddělitelnost kognitivní a podvědomé složky lidského uvažování, a také kognitivní a fyzické části našeho vnímání. Pokud tedy nelze pro výše uvedený účel použít Turingova testu, pak je pro bližší zkoumání lepší soustředit se na úspěšnost v oblasti kategorizace, schopnost a rychlost učení se novým konceptům, umění využití již známých konceptů v novém prostředí, atd. [15]

2.4 Strategické hry

Strategické hry představují robustní prostředí pro metodiky strojového učení. Než se však zaměříme na důvody proč jim věnovat pozornost, podívejme se, jak vypadal vývoj v této oblasti z historického hlediska.

2.4.1 Historický kontext

Již od vzniku moderních počítačů se objevovaly myšlenky, zda může stroj vyrovnat, či dokonce překonat lidskou inteligenci. Její měření však není triviální, a proto se začaly tyto myšlenky ubírat směrem k úkolům, které byly pro člověka složité na uvažování. Již v roce 1950 vyšel článek od amerického matematika Claudea E. Shannona, v němž se zaměřuje na možnost naprogramování počítače k hraní šachu[1] [39]. V práci mimo jiné také odhadl velikost množiny možných odehraných her na 10^{120}.[2] Jím navržený postup na vytvoření programu k hraní šachu byl postaven na metodě *minimax*, kdy využívá aproximační evaluační funkci vztaženou ke stavu (pozicím figurek) hry, zahrnující údaje nejen o počtu figur obou barev na hrací ploše, ale i o strategickém postavení jednotlivých figurek vůči ostatním či hrací ploše. Rozebral ve své práci i výhody a nevýhody čistě logického myšlení oproti lidskému, např. robustnost vůči chybám z nepozornosti, lenosti, či těch pramenících z předpokladu prohry, nebo naopak flexibilitu, představivost a učenlivost člověka.

První program schopný porazit člověka využíval také metody minimax, ale zaměřil se na jednodušší hru známou u nás jako "dáma"a poprvé byl zde ve strategické hře využit koncept učení. Program vytvořený pod záštitou společnosti IBM byl zejména průlomový díky využití metody učení prostřednictvím zobecnění zkušeností z předchozích tahů, čímž efektivně vyřadil nutnost simulovat obrovské množství budoucích tahů. [36] Zároveň zde byl použit způsob učení hraním proti své vlastní předchozí verzi, na čemž je např. postaven níže uvedený AlphaGo Zero. V roce 1957 již existoval první program, který byl schopen šachy hrát [8], a v roce 1958 již vznikl program schopný porazit začátečníka v šachu[3] [31]. Kromě teoretického zájmu několika stovek zasvěcených inženýrů a matematiků se čím dál více projevoval i zájem širšího publika. Publicita tedy byla dalším důvodem, proč zkoumání UI v prostředí strategických her získávalo čím dál více pozornosti u předních technologických firem.

Posuneme-li se na časové ose dále, postupně vznikaly stále sofistikovanější programy pro simulaci her jako jsou dáma [38], šachy [19], Go [49], ale i backgammon[4] [42]. Již zmiňované kolísání zájmu o vývoj UI ale značně komplikovalo pokrok v tomto odvětví. Na skutečně průlomový výsledek přišel čas až v 90. letech, kdy i výkon tehdejších strojů dosahoval dostatečné úrovně. Firma IBM se postarala o sestavení dedikovaného superpočítače pro hraní šachu, jehož program nesl jméno *Deep Blue*. Po prvním duelu v roce 1996 zvítězil Garri Kasparov nad tímto strojem 4-2, o rok později však již Deep Blue poráží ruského šachového velmistra v poměru 3.5-2.5, a v této ikonické hře od té doby první příčky obsazuje umělá inteligence. [10] Než se však i Go dočkalo svého virtuálního šampiona, trvalo to bezmála další dvě dekády. Vítězství počítače *AlphaGo* společnosti Google DeepMind nad světovým šampionem Lee Sedolem v březnu roku 2016 sledovalo 60 milionů lidí [23, 14]. Vzhledem k tomu, že stavový prostor hry Go je tak obrovský, stalo se toto vítězství důležitým milníkem[5] [43].

[1] Tato práce zaměřená čistě teoreticky, bez jakéhokoliv praktického využití. Je však nutno uvést, že Shannon ve své práci uvádí několik možných prakticky využitelných směrů dalšího vývoje postavených na jeho práci.

[2] I v dnešní době se tato hodnota považuje za přesný odhad, a často se 10^{120} označuje jako Shannonovo číslo.

[3] Tento program využíval napodobeninu metodiky, která se stala nepostradatelným jádrem budoucích virtuálních šachových hráčů. Jednalo se o tzv. Alpha-Beta pruning.

[4] U nás tuto hru známe pod názvem vrhcáby.

[5] Hrací deska Go o velikosti 19x19 polí obsahuje přibližně $2.08 * 10^{170}$ legálních stavů, pro porovnání, v pozorovatelném vesmíru je dle odhadů přibližně 10^{80} atomů.

2.4.2 Proč strategické hry?

Strategické hry postupem času upoutávaly pozornost nejen pro svou popularitu, ale i z praktických důvodů, souvisejících s hledáním stále sofistikovanější UI. Dosavadní schopnosti UI lze demonstrovat prostřednictvím strategických her velmi přirozeným způsobem, pochopitelným i pro laika v oblasti informačních technologií. Zároveň se jednalo v případě šachu či Go o přímočaré, pro stroj dobře uchopitelné problémy, a to hlavně z následujících důvodů:

- Jasně definovaná pravidla.
- Jasně definované prostředí.
- Konečný (a dostatečně nízký) počet možných stavů dané hry.

Všechna tři kritéria byla klíčová, jelikož představují dostatečné zjednodušení problémů pro jejich řešení aktuálně dostupnými prostředky a algoritmy. Významným faktorem pro volbu her šachu a Go pro výzkum v této oblasti byla i značně obsáhlá databáze záznamů her získaných díky velké hráčské základně (zejména těch odehraných velmi pokročilými hráči), ze kterých se tyto počítače učily. V případě Deep Blue bylo z hlediska učení ze záznamů ohodnocení tahu ovlivněno několika ukazateli, jako například počtem výskytů daného tahu v záznamech, prestiží hráče odehrávajícího daný tah, výsledkem či komentářem daného tahu, apod. [10] Co se však týče hry Go, nejnovější verze počítače *AlphaGo Zero* představená veřejnosti na podzim roku 2017 se byla schopna učit výhradně hraním proti sobě samotné, přičemž začala od naprosto náhodné hry. Přesto však dosáhla lepších výsledků než jakákoliv předchozí verze AlphaGo, včetně těch, které porazily světové mistry v Go. [40]

Kromě většího stavového prostoru také existují pokusy o řešení komplexnějších problémů prostřednictvím neúplně pozorovatelného stavu. Jako příklad zde můžeme uvést Texas Hold'Em Poker, ve kterém dosáhli agenti *Libratus* [9] a *DeepStack* [29] velmi zajímavých výsledků, přičemž byly schopny při dlouhodobějších hrách jednoduše porážet profesionální hráče ze světové špičky.

Kapitola 3

Agent a strojové učení

Rozebereme si zde nutné teoretické základy pro tvorbu agenta simulujícího hráče takového, jak je popsán v návrhu a implementaci. Specifikace rozhraní pro interakci tohoto agenta s prostředím Starcraft II, které bude cílem trénování agenta, lze nalézt v následující kapitole. Práce a navrhované řešení se zabývá metodami zpětnovazebního učení, uvádím zde tedy související teorii. V práci je využito do značné míry metodik publikovaných teprve velmi nedávno, tyto však vycházejí ze základních principů oboru umělé inteligence. Ačkoli je práce fundamentálně založena na umělých neuronových sítích, nebudu zde jejich princip rozebírat do hloubky a odkáži čtenáře na svou předchozí práci, kde umělé neuronové sítě popisuji detailněji ([45]). Zaměřuji se na dvě klíčové technologie použité v této práci: *Advantage Actor-Critic*, a *Population Based Training*.

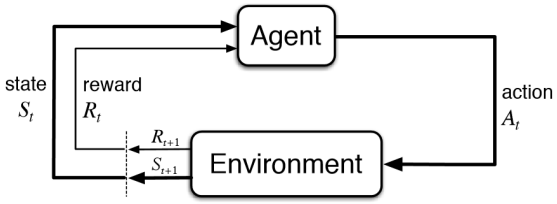
3.1 A3C algoritmus

A3C, neboli *Asynchronous Advantage Actor-Critic* představuje koncepčně i obsahově jednoduchý rámec pro zpětnovazební učení, používající anynchronní metodu gradient descent pro optimalizaci parametrů hlubokých neuronových sítí. [26] U jeho předchůdců se pro učení často využívá ukládání určitého množství provedených kroků, aby bylo možné tyto vzorky dekorelovat v závislosti na čase (např. prostřednictvím náhodného samplingu, či batchování). Následně je lze analyzovat a použít k učení sítě[1]. Pracuje se tedy s jedním agentem běžícím nad jediným prostředím. A3C staví místo toho na paralelizaci skrze spouštění několika agentů paralelně, kde každý z agentů má vlastní instanci prostředí. V této práci je využito rámce *A2C* (*Advantage Actor-Critic*), což není nic jiného než synchronní verze A3C, a tedy ve drtivé většině případů budou informace v této podkapitole platit pro obě verze.

3.1.1 Zpětnovazební učení

Před tím, než se hlouběji pustíme do A3C, by bylo záhodno si krátce ujasnit, co se myslí zpětnovazebním učením (v angličtině reinforcement learning, a dále jej budeme převážně označovat zkratkou RL). U jednoduššího učení s učitelem je pro neuronovou síť vždy k dispozici klíč, jehož hodnotu se snaží síť aproximovat na základě přijatých vstupů. Poté je vyhodnoceno, zda (a případně do jaké míry) výstup neuronové sítě odpovídá očekávanému výstupu, a parametry se odpovídajícím způsobem upraví. U zpětnovazebního učení však

[1] Takovou cestou se ubírá kupříkladu Q-Learning



Obrázek 3.1: Jednoduché schéma popisující interakci agenta využívajícího zpětnovazebného učení s jeho prostředím. Agent provádí akci A_t, načež mu prostředí poskytuje zpětnou vazbu v podobě výsledného stavu S_{t+1} a odměny R_{t+1} Převzato z [17].

žádný takový odpovědní klíč neexistuje, a síť se tedy musí učit jinak. RL agent nemá k dispozici žádná data k trénování, a nezbývá mu nic jiného než učit se za běhu prostřednictvím získávání zkušeností metodou pokus-omyl, přičemž se zaměřuje zejména na dlouhodobé výsledky. To z něj dělá ideálního aktéra v prostředí tzv. *Markovova rozhodovacího procesu* (viz podkapitola 3.1.2).

Formálně lze RL definovat následovně. Mějme prostředí ε, ke kterému je přiřazen agent. Agent může v tomto prostředí pro každý časový krok t zvolit na základě přijatého stavu s_t provedení akce a_t z nějaké množiny možných akcí $\mathcal{A}$ podle strategie π, kde π mapuje stavy s_t k akcím a_t. Jako důsledek provedení akce a_t pak agent získává následující stav s_{t+1} a odměnu r_{t+1}. Tento proces se opakuje, dokud není dosaženo ukončujícího stavu, po kterém je proces restartován. Postupně je takto získána suma získaných odměn $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ v kroku t s využitím diskontního faktoru $\gamma \epsilon (0, 1]$, který určuje, do jaké míry jsou důležité budoucí odměny oproti okamžitým odměnám. Agent si klade za cíl maximalizovat očekávanou odměnu z každého stavu s_t. [26]

3.1.2 Markovův rozhodovací proces

Markovův rozhodovací proces poskytuje matematicky vymezený rámec pro modelování rozhodování v stochastických problémových doménách a prostředích. Tyto procesy jsou velmi užitečné pro studium zpětnovazebného učení. MDP je popsán jako uspořádaná čtveřice $\langle S, A, T, R \rangle$, kde:

- S je konečná množina stavů světa,
- A je konečná množina akcí proveditelných ve světě,
- $T : S \times A \to \Pi(S)$ je přechodová funkce, a jedná se o rozložení pravděpodobnosti přechodu (píšeme $T(s, a, s')$), ze stavu s prostřednictvím akce a do stavu s' a
- $R : S \times A \to \mathbb{R}$ je užitková funkce, vracející okamžitý užitek získaný agentem provedením zvolené akce ve zvoleném stavu (píšeme $R(s, a)$, kde s je zvolený stav, a a je zvolená akce).

V tomto modelu následující stav a očekávaný užitek závisí pouze na aktuálním stavu a prováděné akci; i pokud bychom zohledňovali předchozí stavy, pravděpodobnosti přechodů a očekávaný užitek zůstávají beze změny (což se označuje jako *Markovská vlastnost*, veškeré potřebné informace k zahrnutí historie stavů jsou zakódovány do aktuálního stavu). Svou roli zde má i již zmiňovaný diskontní faktor, zajišťující nejen postupnou degradaci

přidělování odměn za provádění akce v závislosti na čase, ale v důsledku toho i postupnou konvergenci k řešení. V SC2 ovšem nelze z pohledu hráče vypozorovat úplnou informaci o aktuálním stavu, a budeme se tedy zabývat modelem známým jako *částečně pozorovatelný Markovův rozhodovací proces* [2]. Jedná se o zobecnění MDP, které je definováno jako uspořádaná šestice $\langle S, A, T, R, \Omega, O \rangle$, kde:

- S, A, T a R jsou definovány stejně jako u MDP,
- Ω je konečná množina vjemů[3] vnímaných agentem ve světě a
- $O : S \times A \to \Pi(\Omega)$ je vjemová funkce, která pro každou akci a výsledný stav poskytuje rozložení pravděpodobnosti napříč možnými vjemy (píšeme $O(s', a, o)$, tedy jde o funkci pravděpodobnosti, že agent skončí na základě vjemu o a akce a ve stavu s').

POMDP je tedy MDP, která nemá úplnou informaci o aktuálním stavu. Místo toho pozoruje změny stavů na základě provedení akce a výsledného stavu. [21]

3.1.3 Princip A3C algoritmu

Algoritmus staví na faktu, že oproti ostatním RL iteračním metodám založených čistě na odhadu očekávaných výstupů (Q-learning) či jen na strategii (např. Policy Gradient metoda) kombinuje oba přístupy. V případě A3C síť odhaduje jak funkci očekávaných výstupních hodnot (tedy jak užitečné je nacházet se v daném stavu), tak strategii (rozložení pravděpodobnosti nad proveditelnými akcemi), což v síti reprezentují dvě výstupní vrstvy. Oproti předchozím metodám agent lépe a přímočařeji využívá odhad výstupních hodnot ("critic") pro aktualizaci strategie ("actor").[4].

Podíváme-li se tedy již přímo na agenty A3C, kteří byli využiti pro zkoumání schopností RL algoritmů v doméně SC2, zjistíme, že používají hlubokou neuronovou síť o parametrech θ, které definují strategii π_θ. V čase t agent získá vjem s_t, vybere akci a_t s pravděpodobností $\pi_\theta(a_t|s_t)$ a získá ze světa (jeho prostředí) odměnu r_t. Cílem agenta je maximalizace návratové hodnoty $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, kde γ je diskontní faktor[5].

Jedná se o aproximační metodu gradient ascent nad očekávaným výstupem $\mathbb{E}[G_t]$. A3C gradient je pak definován následovně:

$$\underbrace{(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{gradient strategie}} + \beta \underbrace{(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t)}_{\text{gradient aproximace hodnoty}} + \eta \underbrace{\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)}_{\text{regulace entropií}}, \tag{3.1}$$

kde $v_\theta(s)$ je funkce odhadu hodnoty očekávaného výstupu $\mathbb{E}[G_t|s_t = s]$ ze stejné sítě. Místo úplného výstupu pak můžeme ve zmíněném gradientu použít očekávaný výstup skládající se z n kroků $G_t = \sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^n v_\theta(s_{t+n})$, kde n je hyperparametr. Poslední term provádí regulaci směrem k více chaotickému chování, což podporuje prozkoumávání[6]. Hyperparametry β a γ pak upravují důležitost obou faktorů.

[2] POMDP, partially observable Markov decision process.

[3] Vjem zde budeme chápat jako agentův částečně omezený pohled na stav světa.

[4] Proto se algoritmus označuje jako Actor-Critic.

[5] Tento metaparametr určuje, jak klesá hodnota dané události v návaznosti na čas, ve kterém se událost vyskytuje.

[6] Problém exploration vs. exploitation je řešen v podkapitole 3.1.4 , ve zkratce se v tomto případě snažíme vyhnout uváznutí v lokálním maximu.

3.1.4 Problém exploration vs. exploitation

Umělá neuronová síť založená na zpětnovazebném učení se může spoléhat pouze na své počáteční náhodně (nebo jinak) inicializované parametry pro určení další prováděné akce. V důsledku učení se projevuje sklon k opakovanému dosahování neoptimálních výsledků v rámci daného prostředí, jelikož se agent ukotví v lokálním maximu, které je navíc ovlivněno počáteční inicializací. Přístupu k učení tímto způsobem se přezdívá exploitation.

Oproti tomu lze do parametrizace volby další akce zavést chaos. Agent náhodně[7] vybere akci k provedení, čímž prakticky prozkoumává stavy, kterých by jinak daná síť nedosáhla, a akce, které by v daných situacích normálně nepoužila. Tento přístup se označuje jako exploration. Pokud bychom ale vytvořili síť založenou čistě na prozkoumávání, pak taková síť není nijak závislá na svých parametrech, a ty tedy pro ni ztrácejí smysl.

Zřejmě je potřebné oba přístupy nějakým způsobem vyvážit, jelikož mají své klady i zápory. Intuitivně zde můžeme odvodit, že zpočátku musí mít síť více možností pro prozkoumávání stavového a akčního prostoru, jelikož se nemůže spoléhat na své náhodně nainicializované parametry. Postupem času však důležitost parametrů sítě v důsledku učení narůstá, a vliv náhody klesá. Proces učení obecně velmi často takto řeší problém exploration vs. exploitation, a agent musí jen správně korigovat poměr mezi oběma přístupy.

3.2 Population Based Training

Zkoumání konfigurací hyperparametrů je z hlediska časové i výpočetní složitosti velmi náročný problém. Volba je často podmíněna zkušeností, náhodnou volbou, nebo výpočetně náročnými vyhledávacími procesy. Navíc se obecně při procesu trénování volí hyperparametry na začátku, a často zůstávají fixní po celé trvání trénování. *Population Based Training* [20] (zkráceně PBT) je asynchronní optimalizační algoritmus spravující populaci modelů a jejich hyperparametry za účelem maximalizace výkonnosti. Zároveň objevuje nastavení hyperparametrů za běhu, místo držení se fixní konfigurace na začátku procesu trénování.

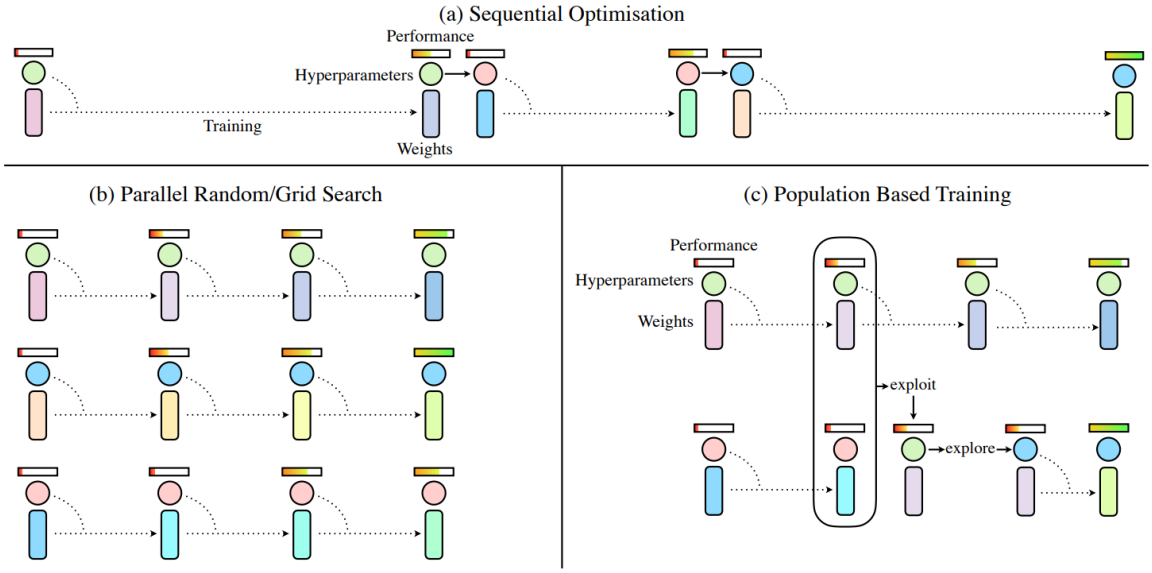
Doposud existovaly dvě hlavní větve manipulace s hyperparametry: paralelní vyhledávání, a sekvenční optimalizace, které mezi sebou vyvažují souběžně používané výpočetní zdroje a čas k dosažení optimálních výsledků. Paralelní vyhledávání spouští mnoho paralelních běhů neuronových sítí (každý z nich s jinými hyperparametry), a vybírá z nich tu nejúspěšnější konfiguraci. Oproti tomu sekvenční optimalizace provádí pár paralelních optimalizačních procesů, ale opakuje tento proces mnohokrát sekvenčně, aby postupně předávala informace z předešlých běhů pozdějším, čímž obecně dojde k nejlepším výsledkům, ale pro dlouhé procesy je prakticky příliš náročná.

PBT z pohledu časové náročnosti nevyžaduje více než jeden optimalizační proces, nepotřebuje sekvenční běhy, a je i výkonnostně úspornější než paralelní metody jako např. grid search [20]. Vizualizace procesu hledání hyperparametrů je na obrázku 3.2.

V případě aplikace této technologie na algoritmus A2C jde to optimalizaci parametrů θ modelu f vedoucí k maximalizaci očekávané návratové hodnoty episody $\mathbb{E}_\pi[R]$, kde π je strategie. Vzhledem k tomu, že modelem je pro nás neuronová síť, optimalizujeme iterativně váhy θ použitím SGD[8] nad uvedenou cílovou funkcí. Každý krok této iterativní optimalizační procedury `step` aktualizuje parametry modelu, a je sám závislý na některých z parametrů $h \epsilon \mathcal{H}$ (často označovaných jako hyperparametry).

Iterace parametrických aktualizačních kroků:

[7] Náhoda může být zavedena např. naivně přímo do volby akce, či manipulací s parametry výběru.
[8] stochastic gradient descent



Obrázek 3.2: Obvyklá paradigmata hledání konfigurací hyperparametrů: sekvenční optimalizace a paralelní vyhledávání, porovnaná s metodou PBT. (a) Sekvenční optimalizace vyžaduje několik běhů trénování k dokončení (potenciálně s předčasným ukončením), po čemž jsou zvoleny nové hyperparametry, a model je s nimi přetrénován znovu od začátku. Jedná se o inherentně sekvenční proces, který vede k dlouhému trvání optimalizace, ale minimálnímu využití výpočetních zdrojů. (b) Paralelní random/grid search trénuje více modelů paralelně s různými počátečními konfiguracemi hyperparametrů a vah s tím, že jeden z modelů bude nejlépe optimalizován. Vyžaduje pouze jeden běh, ale potřebuje mnohem více výpočetních zdrojů pro běh několika modelů paralelně. (c) PBT začíná jako paralelní algoritmus s náhodně nainicializovanými vahami a hyperparametry. Každý běh však periodicky vyhodnocuje svou výkonnost. Pokud model zjistí, že jeho výkonnost je nedostačující, využije zbytku populace nahrazením sebe sama výkonnějším modelem, a prohledá lepší kombinaci hyperparametrů, než opět začne s trénováním. Tento proces umožňuje online optimalizaci hyperparametrů, a zaměřuje výpočetní zdroje na hyperparametrický a váhový prostor, který má nejvyšší šanci vyprodukovat kvalitní výsledky. Převzato z [20].

$$\theta \leftarrow \texttt{step}(\theta|h)$$

jsou zřetězeny do sekvence aktualizací, které v ideálním případě konvergují k optimálnímu řešení:

$$\theta^* = \texttt{optimise}(\theta|\boldsymbol{h}) = \texttt{optimise}(\theta|(h_t)_{t=1}^T) = \texttt{step}(\texttt{step}(...\texttt{step}(\theta|h_1)...|h_{T-1})|h_T). \quad (3.2)$$

Nejen že tento proces je výpočetně náročný, ale řešení je také typicky velmi citlivé na výběr sekvence hyperparametrů $\boldsymbol{h}$, což může vést až k neúspěchu dosažení konvergence při optimalizaci θ. Správná volba hyperparametrů je velmi složitá, a vzhledem k jejich vazbě na iterační krok roste počet možných hodnot exponenciálně v závislosti na čase. V důsledku je třeba vyzkoušet mnoho kombinací hodnot hyperparametrů $\boldsymbol{h}$ prohledáváním:

$$\theta^* = \texttt{optimise}(\theta|\boldsymbol{h}^*), \boldsymbol{h}^* = \underset{\boldsymbol{h}\epsilon\mathcal{H}^T}{\arg\max}\ \texttt{eval}(\texttt{optimise}(\theta|\boldsymbol{h})). \quad (3.3)$$

PBT navrhuje trénování N modelů $\left\{\theta^i\right\}_{i=1}^N$ tvořící populaci $\mathcal{P}$, kde optimalizujeme každý model s jinými počátečními hyperparametry $\left\{\boldsymbol{h}^i\right\}_{i=1}^N$. Cílem je tedy nalezení optimálního modelu napříč celou populací $\mathcal{P}$. Místo využití přístupu paralelního prohledávání však využijeme částečná řešení v populaci k provádění meta-optimalizace, kde adaptujeme hyperparametry h a váhy θ na základě úspěšnosti celé populace. Tato meta-optimalizace je prováděna prostřednictvím metod `exploit` a `explore`. Metoda `exploit` zajistí, že při nedostatečném výkonu modelu nahradíme jeho váhy a hyperparametry modelem s nejvyšší výkonností a poté `explore` zavede do hodnot hyperparametrů šum. [20] Tímto postupem je zajištěno, že populace prozkoumává zajímavé sektory hyperparametrického a váhového prostoru.

Kapitola 4

Starcraft II

Následujícím milníkem, kde může umělá inteligence hledat způsob pokoření svého lidského protějška se stává počítačová hra s názvem *Starcraft II*. Stejně jako jeho předchůdce je SC2 real-time[1] strategická hra vydaná společností Blizzard Entertainment. Přestože byla hra vydána v roce 2010, tak si i po několika letech udržuje své fanoušky (od konce roku 2017 je dokonce zdarma). V této kapitole si přiblížíme principy a pravidla této hry. SC2 má i režim pro jediného hráče, ale pro nás je z hlediska SU zajímavý hlavně mód pro více hráčů.

4.1 Proč Starcraft II?

Existuje hned několik důvodů, proč je SC2 vhodným kandidátem pro výzkum v této oblasti:

- Velká hráčská základna,

SC2 a jeho předchůdce se těší oblíbenosti hráčů již dohromady přes 20 let. Byl uspořádán nespočet turnajů i s finančními odměnami pro vítěze, a dokonce existují i turnaje výhradně mezi UI, prozatím alespoň v první verzi Starcraftu. Popularita SC2 se pak v důsledku odráží i na počtu záznamů odehraných her a hráčů vysoké úrovně.

- jasně definovaná pravidla,

Je vždy jasně definováno jaké akce může hráč v dané situaci provést, a zároveň je i jasně dáno, jaké jsou podmínky pro vítězství.

- jedná se o více-agentní problém[2],

V SC2 spolu může soupeřit i více hráčů najednou o suroviny a vliv na herním poli. Zároveň se jedná o více-agentní problém i na nižší úrovni, jelikož každý hráč ovládá až stovky jednotek, které spolu musí spolupracovat za účelem plnění společného cíle.

- složitost hry jako takové,

Přestože je cíl hry jasně daný, jeho dosažení je značně komplikované. Aby hráč dosáhl vítězství, musí zároveň průběžně provádět několik podúkolů, a vyvažovat pozornost a prostředky mezi nimi. Těmito podúkoly jsou např. sběr surovin, průzkum, nebo stavba jednotek.

[1] Tedy hráči spolu soupeří v reálném čase bez omezení počtu a pořadí provádění akcí, střídání tahů zde vůbec nefiguruje.

[2] Více agentů v rámci jedné hry. SC2 umožňuje začít hru až s 8 hráči.

- neúplná znalost stavu hry,

Hráč je v drtivé většině situací omezen aktuální pozicí kamery, která musí být aktivně přesunována, čímž lze získávat informace o stavu. Navíc zde funguje mechanismus zvaný "fog-of-war", který skrývá neprozkoumané části herní mapy. Hráč musí tuto mapu aktivně prozkoumávat, aby zjistil stav svého protivníka.

- obsáhlá a rozmanitá množina akcí,

Hráč vybírá akci z obrovského množství možných kombinací (až 10^8 možností) prostřednictvím point-and-click rozhraní. Mnoho unikátních jednotek má své vlastní akce. Navíc se dostupnost akcí může v průběhu hry měnit na základě zkoumání technologií hráčem.

- délka hry,

Jedna úplná hra může trvat i mnoho tisíc snímků, a hráč musí provádět akce, jejichž důsledky mohou být znatelné až mnohem později.

4.2 Základní princip

Základním cílem každého hráče je vybudovat armádu pro získání nadvlády na mapě, a zničit všechny jednotky svého protivníka. Hra je situována na jedné z vybraných předdefinovaných map, a hráč ji vidí z isometrického pohledu. Před začátkem si každý z hráčů vybere jednu ze tří možných ras - Terran, Zerg a Protoss. Tato volba určuje, jaké jednotky, budovy a technologie budou k dispozici. Po započetí hry mají hráči k dispozici vlastní hlavní budovu umístěnou u jednoho z nalezišť surovin, a několik jednotek typu Worker (dělník). Tito dělníci jsou pak schopni jak sběru surovin, tak stavby a oprav budov. Suroviny lze pak použít právě pro výstavbu budov (různé budovy mají různou funkci), pro vytvoření dělníků nebo jiných jednotek, a zkoumání technologií. Počáteční stav je ukázán na obrázku 4.1.

4.3 Zjišťování aktuálního stavu

Jak již bylo uvedeno v předchozí části, hráč zpravidla nemá úplný přehled o stavu hry. Náhled na dění je omezen isometrickou kamerou, zobrazující pouze malý zlomek celkového stavu. Tuto kameru je však možné libovolně přesunovat (např. prostřednictvím mini-mapy) v rámci dané mapy. Mnohem více však hráče omezuje výše zmíněný mechanismus "fog-of-war". Ve výchozím stavu jsou zobrazeny pouze informace o terénu a umístění surovinových nalezišť. Každá vlastní jednotka (resp. budova, příp. i některá akce/technologie) disponuje rádiem viditelnosti, který odhalí všechny jednotky a budovy v dosahu[3]. Pokud jsou odhaleny budovy, tak je v případě ztráty viditelnosti zaznamenán jejich poslední stav[4]. Algoritmus SU se s tímto faktem tedy musí vyrovnat, a to nejen aktivním zkoumáním neprozkoumaných části herní mapy, ale i rozhodováním na základě neúplných znalostí o stavu. Na strategické myšlení hráčů má vliv také výběr konkrétní mapy, na které bude hra probíhat. Každá mapa disponuje několika důležitými vlastnostmi:

[3] Výjimku však tvoří jednotky, které jsou ve skrytém režimu.
[4] Například úroveň poškození, nebo úplné zničení.

- Velikost mapy

Do značné míry určuje počet surovinových nalezišť, a také ztěžuje hledání protivníkových jednotek nebo základen. Tento fakt může fundamentálně změnit optimální taktiku, a to hlavně co se týče vytváření jednotek.

- Členitost terénu

Každé místo na mapě je položeno na jedné ze dvou úrovní - níže či výše. Toto rozdělení omezuje viditelnost pozemních jednotek.

- Surovinová naleziště

Velikost a počet surovinových nalezišť např. určuje, jak rychle by měl hráč expandovat při stavění dalších základen, nebo jaké množství surovin investovat do obrany jednotlivých základen.

- Nedostupná místa

Pro většinu jednotek nedosažitelné části mapy. Výjimku mohou tvořit létající jednotky.
Všechny tyto faktory ovlivňují optimální řešení pro daný zápas. Počítač, který chce zvítězit nad lidským protějškem, musí vše zde zmíněné vzít v potaz, jelikož i průměrný hráč těchto faktorů dokáže využít ve svůj prospěch.

Obrázek 4.1: Snímek herní obrazovky na začátku hry z pohledu jednoho z hráčů. Uprostřed je vidět hlavní budova rasy Terran, a několik dělníků již aktivně těžících suroviny. Modré krystaly (mineral shards) jsou první surovinou, zelený plyn (vespene gas) jsou druhou. Vpravo nahoře je vidět popořadě aktuální množství nasbíraných krystalů, plynu a naplnění a velikost populačního limitu. Vlevo dole se na obrazovce nachází tzv. mini-mapa (zjednodušený pohled na aktuální stav celé mapy), uprostřed spodní části se zobrazuje přehled vybraných budov/jednotek (a výrobní fronta), a vpravo dole je kontextové menu zobrazující akce proveditelné vybranou jednotkou/budovou. Viditelná část mini-mapy je vyznačena světle, neprozkoumaná část tmavě. Světlemodré body jsou naleziště surovin, zeleně (barvu si hráč volí sám před začátkem hry) jsou vyznačeny budovy a jednotky hráče. Bílý útvar pak ukazuje aktuální pozici kamery ve vztahu k mini-mapě.

Kapitola 5

Rozhraní mezi agentem a prostředím

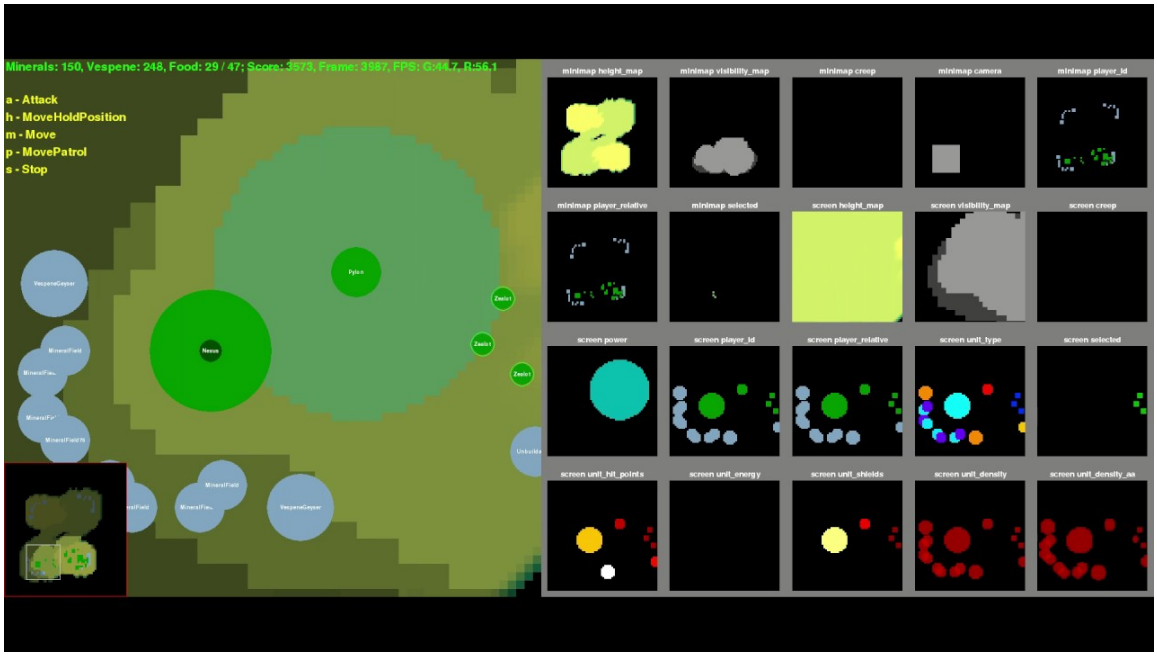
Nyní si vysvětlíme, jakým způsobem je možné propojit našeho agenta (virtuálního hráče, neboli algoritmus strojového učení) se samotnou hrou. K napojení je použito API pro strojové učení vyvinuté samotným vývojářem hry [3], které poskytuje ostatním vývojářům napojení do běhu SC2 i mimo standardní uživatelské rozhraní. Zároveň je použita nástrojová sada *PySC2*, která usnadňuje použití tzv. *feature-layer* části API pro vytvořeného agenta [2].

Umělá inteligence je o praktickém uvažování, a to takovému, které vede k nějakému činu. Spojením vnímání, uvažování a konání v určitém prostředí vzniká agent [34]. Fázi uvažování lze provádět bez napojení na prostředí. Podívejme se tedy na to, jakým způsobem agent vnímá prostředí SC2, a jak je v něm schopen reagovat výběrem akce. Je nutno uvést, že se zde zaměřuji spíše na vysvětlení rozhraní, formálnější popis je k nalezení v [44].

5.1 Fáze vnímání

Hra používá herní engine k vykreslování scény do 3-dimenzionálního obrazu, který je pak přenesen na 2D obrazovku a prezentován uživateli. Tato obrazovka je však pro člověka snadno čitelná, jelikož se nám jeví jako scéna z reálného světa[1]. Stroj začínající v kontextu neuronových sítí prakticky jako tabula rasa však nerozumí skutečnému světu, a pro účely hraní SC2 je pro něj toto vnímání značně obskurní a komplexní. Přicházíme tedy do styku se zmiňovaným feature-layer API. Toto prostředí místo vykreslování 3D obrazovky rozděluje její důležité prvky do několika vrstev, které jsou samostatně pro stroj mnohem čitelnější, a lépe zpracovatelné. Jsou oproštěny o většinu nadbytečných vizuálních jevů (3D prostor, speciální vizuální efekty, apod.), přičemž stále poskytují ucelený pohled na aktuální stav hry. Další zjednodušení spočívá v zmenšení rozlišení těchto vrstev, jelikož čistě funkční, minimalistický pohled představuje mnohem efektivnější způsob získávání informací o hře (a stroj "neocení"estetický vzhled hry). Vizuální interpretaci tohoto API lze vidět na obrázku 5.1.

[1] Samozřejmě bráno ve smyslu vnímání světa jako 3D interaktivního prostředí, perspektiva je pro lidské chápání přirozený jev. Nebavíme se zde o zasazení zobrazovaného fiktivního světa do skutečnosti, i když je to určitě zajímavá myšlenka.



Obrázek 5.1: PySC2 feature-layer API. Prakticky se jedná o snímek herní situace v průběhu hry, podobně jako u obrázku 4.1, avšak nyní z pohledu tohoto API. Jednotlivé popisky nad obrázky v API vyznačují, kterou vlastnost daný obrázek zachycuje. Rozlišení jednotlivých obrázků pro účely dalšího zpracování je možné v rámci tohoto rozhraní měnit. Jsou také k dispozici důležité skaláry (množství natěžených surovin, aj.), a množina proveditelných akcí.

Jednotlivé vrstvy poskytují více či méně užitečné, druhově odlišné informace. Jedná se kupříkladu o:

- Typ jednotek,
- vlastnictví jednotek,
- viditelnost mapy,
- výškovou mapu,
- pozici kamery na minimapě,
- a další.

Pro korektní vyhodnocení optimální akce je třeba efektivní analýza výsledků fáze pozorování. Agentovi jsou stále poskytovány veškeré vypozorované informace, a ten sám zodpovídá za jejich filtrování a zpracování.

5.2 Fáze akce

Hráči SC2, umístění na vysokých pozicích žebříčku, musejí vstřebávat enormní množství informací najednou a provádět úkony vysokou rychlostí (profesionální hráči dokážou vyko-

návat až *500 APM*[2]) [44]. Základní množina všech akcí čítá 524 úkonů, přičemž velká část z nich vyžaduje alespoň bod buď na hlavní obrazovce, nebo na mini-mapě. Pokud bychom transformovali akční prostor do jedné dimenze, čítal by výsledný vektor přes 100 milionů akcí. Agent může prostřednictvím API za jakékoliv herní situace zažádat o provedení libovolné akce, avšak musí kontrolovat, zda je zvolená akce v aktuálním kontextu validní. Díky svému výpočetnímu výkonu dokáže samozřejmě provádět mnohem více akcí, než člověk. Cílem ale není dokázat, že agent koná rychleji než člověk. Stejně jako u práce Google DeepMind tedy omezíme počet akcí tak, že povolíme provádění akce jednou za 8 herních snímků, což je ekvivalentní přibližně 180 APM. Takové množství akcí je schopen provádět středně pokročilý hráč SC2. [44] Zároveň tento postup aplikují i ostatní implementace SU, a tedy by měly teoreticky být porovnatelné s vlastní implementací navrhovanou touto prací.

[2]Actions Per Minute, úkonů za sekundu, do kterých se nepočítá akce výběr jednotky/jednotek.

Kapitola 6

Dosavadní implementace agentů strojového učení

Z předchozích kapitol je jistě zřejmé, že hra Starcraft II představuje velmi komplexní problém pro SU, ještě mnohem komplexnější než hra Go. To jen podtrhuje fakt, že žádná dosavadní implementace založená čistě na SU nedokázala sama o sobě porazit ani hráče-amatéra v plné hře SC2. V důsledku bylo vytvořeno i několik mini-her v prostředí SC2, jejichž komplexnost je sice různá, přesto však všeobecně mnohem menší, než je tomu u plné hry. Tím je umožněno prozkoumat izolovaně jednotlivé části hry s jednodušším a jasnějším systémem hodnocení.

Jedná se o velmi aktuální a novou problematiku, prakticky téměř všechny pokusy o vytvoření agenta pro plnou hru končí skriptovaným chováním bota s nějakou vložkou v podobě RL komponenty (např. [47, 33, 41]) na jejíž vyhodnocení se práce zaměřuje. Existují i analýzy zaměřující se pouze na analytický popis strategií v SC2 ([18]) nebo na tvorbu datasetu pro učení RL agentů v prostředí SC2LE ([46]).

Nyní krátce rozebereme trojici doposud nejúspěšnějších agentů (uveřejněných do konce roku 2017) implementovaných společností Google DeepMind, demonstrující schopnost RL[1] agentů vyvozovat strategie a úspěšně řešit tyto mini-hry. Těmito agenty jsou *Atari-net*, *FullyConv* a *FullyConv LSTM*.

6.1 Architektury jednotlivých agentů

Tyto již zavedené architektury agentů byly převzaty z literatury, a byly adaptovány společností Google DeepMind tak, aby vyhovovaly specifikům SC2, a to zejména z pohledu akčního prostoru [44, 27, 26]. Zmínění agenti používají algoritmus A3C (Asynchronous Advantage Actor Critic) tak, jak je popsán v kapitole 3.1.3. Všechny později uvedené výsledky byly získány prostřednictvím prostředí PySC2, a tedy budou použity k porovnání s vlastním řešením.

6.1.1 Atari-net

Tato adaptace vychází z architektury úspěšně použité pro Atari benchmark [7] a prostředí Deep Mind Lab [6]. Zpracovává vjemy z kamery a minimapy stejnou konvoluční sítí jako v [26] - dvě vrstvy s filtry 16,32 o velikosti 8,4 a krocích délky 4, resp. 2. Neprostorové

[1] RL = reinforcement learning, neboli zpětnovazební učení.

vlastnosti jsou zpracovávány lineární vrstvou s aktivační funkcí tanh. Výsledky jsou spojeny a předány lineární vrstvě s ReLU aktivací. Výsledný vektor je pak použit jako vstup pro lineární vrstvy, které nezávisle poskytují strategie pro identifikátor funkce a^0 a každý funkční argument akce $\{a^l\}_{l=0}^{L}$. Atari-net byl tedy revoluční ve svém přístupu k učením se strategiím prostřednictvím zpětnovazebného učení přímo ze vstupu získaného z kamery.

6.1.2 FullyConv

Tento agent se skládá z plně konvoluční neuronové sítě, která předvídá prostorové akce přímo skrze posloupnost konvolučních vrstev, které zachovávají prostorovou informaci díky zarovnávání v každé z jednotlivých vrstev. Implementace přijímá zvlášť vjemy z kamery a mini-mapy přes oddělené dvouvrstvé konvoluční sítě s filtry 16,32 o velikosti 5×5, resp. 3×3. Reprezentace stavu se pak skládá ze spojení výstupu sítě kamery, výstupu sítě mini-mapy a vektorových statistik ze hry.

V případě provádění neprostorových akcí je pro výpočet strategie výsledný stav předán plně propojené vrstvě o 256 jednotkách s ReLU aktivacemi, následované plně propojenými lineárními vrstvami. Strategie nad prostorovými akcemi je ze stavu získána použitím konvolučního filtru o velikosti 1×1 s jediným výstupním kanálem. Oproti Atari-net nejdůležitějším rozdílem zůstává tedy zachovávání relativní prostorové informace i napříč jednotlivými vrstvami.

6.1.3 FullyConv LSTM

Předchozí implementace je čistě dopředná, a nefiguruje v ní paměť. Tato modifikace FullyConv agenta pouze přidává konvoluční LSTM modul poté, co jsou spojeny výstupy sítí kamery a mini-mapy a informace z vektoru statistik.

6.2 Úspěšnost agentů

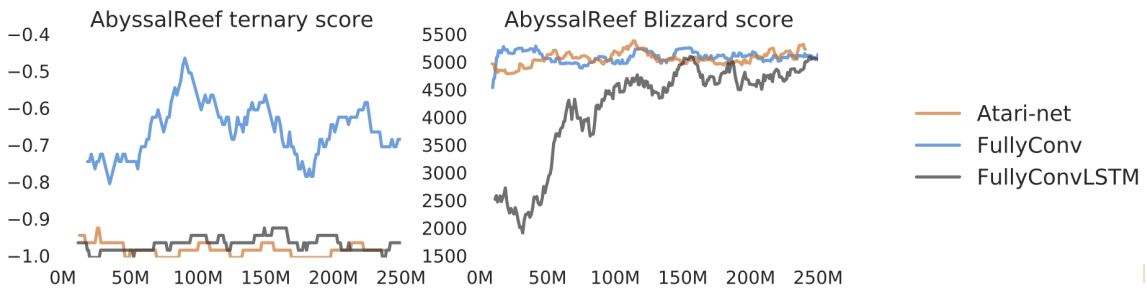
Bližší popis nastavení hyperparametrů sítí a detailnější popis experimentů lze nalézt v článku [44]. Nejdříve shrneme jejich úspěšnost v rámci celé hry, a pak se podíváme na mini-hry v rámci SC2.

6.2.1 Plná hra

Byla vybrána jedna z map používaných v oficiálních zápasech na žebříčku SC2 s názvem Abyssal Reef LE. Agent soupeřil proti nejslabší vestavěné umělé inteligenci v zápase Terran vs. Terran. Maximální délka hry byla stanovena na 30 minut, po čemž byla vyhlášena remíza, a hra skončila.

Výsledky experimentů jsou uvedeny na obrázku 8.4. Žádný z agentů se nedokázal naučit vítěznou strategii k vítězství v plné hře. Nejúspěšnější agent pouze oddaloval prohru využitím Terranské schopnosti vyzdvihnout budovy a přesunout je z dosahu nepřátel. Agenti trénovaní prostřednictvím nativního skóre uvnitř hry se pouze vyhýbali akcím přerušujícím těžbu minerálů, čímž si zajišťovali stabilní růst skóre. Neodhodlali se vůbec ke stavbě budov ani jiných jednotek.

Pro plnou hru existuje velké množství volně dostupných záznamů z žebříčku SC2, které byly také využity pro trénování. Modely postavené na zmíněných modelech a trénované na těchto záznamech nezaznamenaly sice vítězství v plné hře, ale více se snažily imito-



Obrázek 6.1: Výkon všech tří agentů v rámci plné hry proti nejjednodušší verzi vestavěného AI oponenta. V prvním grafu je jako odměna při učení poskytován výsledek celé hry (-1 = prohra, 0 = remíza, 1 = výhra), ve druhém je použito nativní skóre, jak je implementováno společností Blizzard. Žádný agent nedokázal zvítězit ani v jedné hře.

vat samotné hráče - dostaly se ke stavbě produkčních budov a jednotek, ale žádné konkrétní strategie nedosáhly. Dokonce byly schopny dosáhnout lepších výsledků u mini-hry `BuildMarines` než modely bez učení se záznamy. Z pokroku oproti modelům netrénovaných na záznamech je však zřejmé, že tato cesta může vést k úspěšnému řešení problému.

6.2.2 Mini-hry

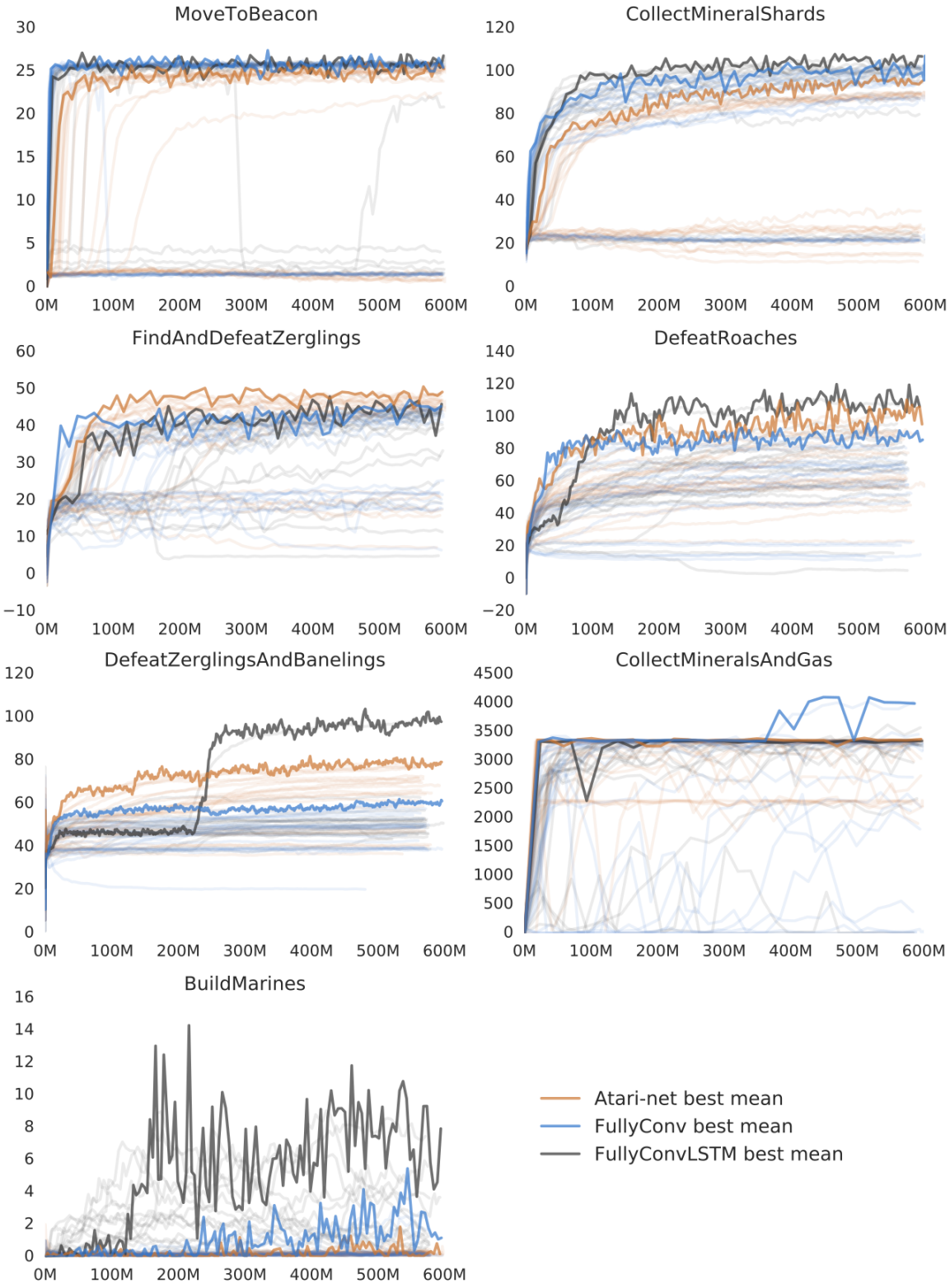
K prozkoumání částí hry v izolaci bylo sestaveno několik mini-her. Jedná se o jednoduché scénáře na malých mapách, které byly vybudovány s cílem otestovat sítě při mnohem menším počtu možných akcí a při provádění úkolů s mnohem jasnější strukturou odměňování. Těchto 7 mini-her je definováno následovně:

- `MoveToBeacon`: Agent ovládá jedinou jednotku Marine, která obdrží +1 skóre pokaždé, když dorazí k vyznačenému bodu. Jedná se o základní test s triviální strategií.
- `CollectMineralShards`: Agent má k dispozici dvě jednotky Marine, a musí je zvolit a sbírat s nimi minerály. Čím efektivněji pohybuje s jednotkami, tím vyššího skóre dosáhne.
- `FindAndDefeatZerglings`: Agent začíná s třemi jednotkami Marine, a musí prozkoumávat mapu, najít a ničit jednotky Zergling. Zahrnuje průzkum a pohyb kamery.
- `DefeatRoaches`: Agent začne s 9 jednotkami Marine, a musí porazit 4 jednotky Roach. Pokaždé, když porazí všechny jednotky Roach, je mu jako odměna přidáno 5 jednotek Marine, a objeví se 4 nové jednotky Roach. Obdrží 10 bodů za zničení jednotky Roach, odebrán 1 bod za smrt jednotky Marine.
- `DefeatZerglingsAndBanelings`: Podobná jako předchozí mini-hra, ale místo jednotek Roach se objevují jednotky Zergling a Baneling. Vyžaduje rozdílnou strategii, protože jednotky mají jiné schopnosti.
- `CollectMineralsAndGas`: Agent začíná s malou základnou, a je odměněn za sběr surovin v omezeném čase. Úspěšný agent musí stavět nové pracovníky pro sběr více surovin.
- `BuildMarines`: Opět začátek s malou základnou, agent odměňován za stavění jednotek Marine. Již je třeba budovat větší množství budov a zároveň udržovat stálý přísun

surovin. Akční prostor je limitován na minimální počet akcí, který je k tomuto úkolu potřeba.

Veškeré mini-hry mají fixní časový limit, a jsou blíže popsány v [44].

Výsledky agentů byly porovnány s dvěma lidskými hráči: amatérským hráčem z Google Deep Mind a mistrem z oficiálního žebříčku. Všichni agenti dosahovaly suboptimálních výsledků v porovnání s mistrem, s jedinou výjimkou nejjednodušší mini-hry `MoveToBeacon`, která vyžaduje pouze rychlé reakce, kde se očekává vítězství RL agentů. V některých mini-hrách, jako je `DefeatRoaches` a `FindAndDefeatZerglings`, dosáhli agenti dobrých výsledků v porovnání s hráčem-amatérem. Detailní rozpis skóre jednotlivých subjektů je uvedeno v [44].



Obrázek 6.2: Trénovací proces pro uvedené architektury agentů. Samotné linie představují průměrné skóre při plnění jednotlivých mini-her jako funkci počtu herních kroků. Slabé linie vyznačují celkově provedených 100 pokusů o trénování každého z agentů s různými hyperparametry, tučně vyznačená linie je vybraný nejlepší průběh. Linie byly vyhlazeny pro lepší čitelnost. Převzato z [44].

Kapitola 7

Návrh a implementace vlastního řešení

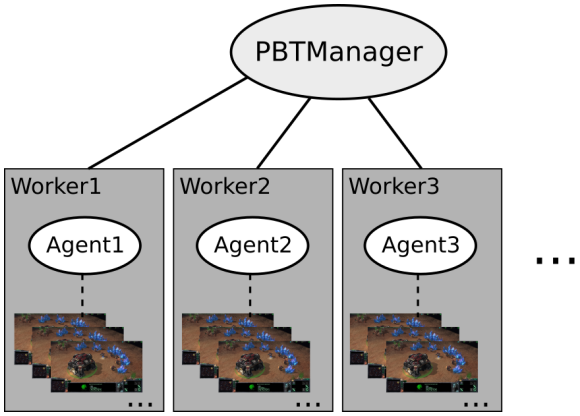
Z výše uvedených výsledků můžeme jednoznačně usoudit, že dosavadní agenti zdaleka nedosahují schopností lidského hráče v plné hře SC2. I u složitějších mini-her lze vidět značné nedostatky ve výkonnosti. Nicméně problematika se nachází v relativně raném stádiu, samotná nástrojová sada byla vydána teprve v srpnu roku 2017 společně s uvedenými agenty, kteří, přestože pochází z dílny jednoho z gigantů na poli strojového učení, jsou stále považováni jen za výchozí agenty pro testování funkcionality API. Přesto představuje překonání těchto milníků velmi složitý úkol, a jakékoliv zlepšení oproti předchozím výsledkům by představovalo pokrok.

Předchozí práce byly výsledky dlouhodobé práce vývojového týmu pracující se state-of-the-art technologiemi a velmi silným HW zázemím. Navrhuji zde proto řešení propojující upravenou verzi algoritmu A3C zvanou A2C s optimalizačním algoritmem PBT (Population Based Training). A2C by měl sloužit jako správný odrazový můstek, poněvadž vychází právě z předchozí práce týmu Google Deep Mind. Posouváme však možnosti zrychlením hledání hyperparametrů prostřednictvím paralelizace a postupné optimalizace použitím algoritmu PBT.

7.1 Implementační prostředí

Implementace, následné testování a provádění experimentů probíhá na operačním systému *Linux*, konkrétně na distribuci *Ubuntu* verze 16.04. Jako programovací jazyk je zvolen *Python 3.5*, jelikož prostředí *PySC2* je implementováno právě v tomto jazyce. K vytvoření a správě umělé neuronové sítě využívám knihovny *tensorflow* a *tensorflow-gpu*. Pro implementaci samotného algoritmu autor vychází kromě jiného zejména z práce publikované společností DeepMind pro uvedení alternativního způsobu přistupování k machine learning API pro Starcraft II prostřednictvím PySC2 (tedy pythonovského prostředí pro posilované učení určeného specificky pro SC2) [2], práce a obecné open source implementace A2C vydaného společností OpenAI [48, 1], práce uvádějící Population Based Training metodu jako alternativní způsob hledání optimálního řešení v hyperparametrickém a váhovém prostoru oproti sekvenčnímu a paralelnímu prohledávání [20], referenčního bota vytvořeného v PySC2 prostředí [32], nebo také již zmiňované knihovny tensorflow/tensorflow-gpu a jejích manuálových stránek [4].

Hardwarové specifikace strojů použitých k testování pak naleznete v kapitole 8.



Obrázek 7.1: Zjednodušený konceptuální obrázek hierarchie v rámci použité architektury. PBTManager spravuje populaci modelů (workerů) a komunikuje s nimi, zatímco modely samy spravují a komunikují s jim přidělenými běžícími prostředími SC2, kde probíhají simulace her a mini-her. Pro detailnější ilustraci prostřednictvím diagramu tříd nahlédněte do obrázku 7.3.

7.2 Advantage Actor Critic

A2C neboli Advantage Actor Critic je čistě synchronní, deterministická verze algoritmu A3C popsaném poprvé v [26]. Doposud nebylo prokázáno (podle mého nejlepšího vědomí na základě hledání relevantních zdrojů), že zavedení asynchronního přístupu má jakýkoliv vliv na úspěšnost algoritmu, a jednalo se pouze o implementační detail, který umožňoval rychlejší trénování implementací založených na CPU. V srpnu tohoto roku vydala společnost OpenAI svou open-source implementaci algoritmu A2C, která je tedy volně použitelná pro veřejnost. Tento algoritmus je tedy využit a adaptován pro SC2. [48]

7.3 Návrh architektury

PBT je přístup založený na správě a vyhodnocování paralelně běžících modelů, blíže popsaný v podkapitole 3.2. V této práci nalézá použití při optimalizaci hyperparametrů umělé neuronové sítě využívané pro simulaci rozhodovací složky hraní SC2. PBT hraje klíčovou roli napříč celým návrhem.

Architekturu tedy můžeme hierarchicky rozdělit do tří celků (tříd), jež bude třeba brát v úvahu, od nejvyšší úrovně po nejnižší:

- `PBTManager`, spravující populaci modelů.
- `Worker`, pracuje nad konkrétním modelem a skupinou prostředí `SC2Env`.
- `SC2Env`, simulační prostředí instance hry Starcraft II.

Podíváme se tedy nejdříve na funkcionalitu jednotlivých částí, načež si v závěru ujasníme, jak do sebe všechny části zapadají.

7.3.1 Třída PBTManager

`PBTManager` zodpovídá za vytváření, porovnávání, vyhodnocování a mazání modelů, a zároveň se stará o jejich komunikaci a přiřazování simulačních prostředí. Manažer je řízen z hlavního procesu, a na základě konfiguračního souboru `config.py` vytváří modely a deleguje jim zodpovědnost za jím přiřazený počet simulačních prostředí.

Nejdříve se shromáždí všechny potřebné informace z konfiguračního souboru:

- Populační parametry - počet trénovaných modelů, počet prostředí přiřazených k modelům, aj.,
- Parametry prostředí - volba konkrétní mapy/minihry, počet herních kroků na jednu akci agenta, rozlišení feature vrstev pro herní obrazovku a minimapu, aj.,
- Síťové parametry - Discount parametr, distribuce inicializačních hodnot vah, nastavení hodnoty normalizačního gradientu, aj.,
- PBT parametry - stanovení počtu batchů před evaluací jednotlivých modelů, metriky pro vyhodnocování,
- Dávkovací parametry - velikost batchů, počet vyhodnocovaných epizod před ukončením trénování, frekvence logování výsledků.

Na základě těchto parametrů vytvoří `PBTManager` $k \times n$ prostředí, kde k označuje počet vytvářených modelů (které zapouzdřuje třída `Worker`), a n počet prostředí přiřazených k jednomu modelu, a seskupí je do k skupin. Každý model je vytvořen a veden jako podproces hlavního procesu s využitím pythonovské knihovny `multithreading`. Jakmile je vytvořena populace modelů o požadovaném počtu (k), a je jim přiřazen stanovený počet prostředí (n, neboli jedna celá skupina), stává se hlavní činností manažera obsluha běžících procesů prostřednictvím `Pipe` spojení. `PBTManager` pracuje skutečně jako server očekávající zprávy od jednotlivých modelů a obsluhující požadavky podle potřeby.

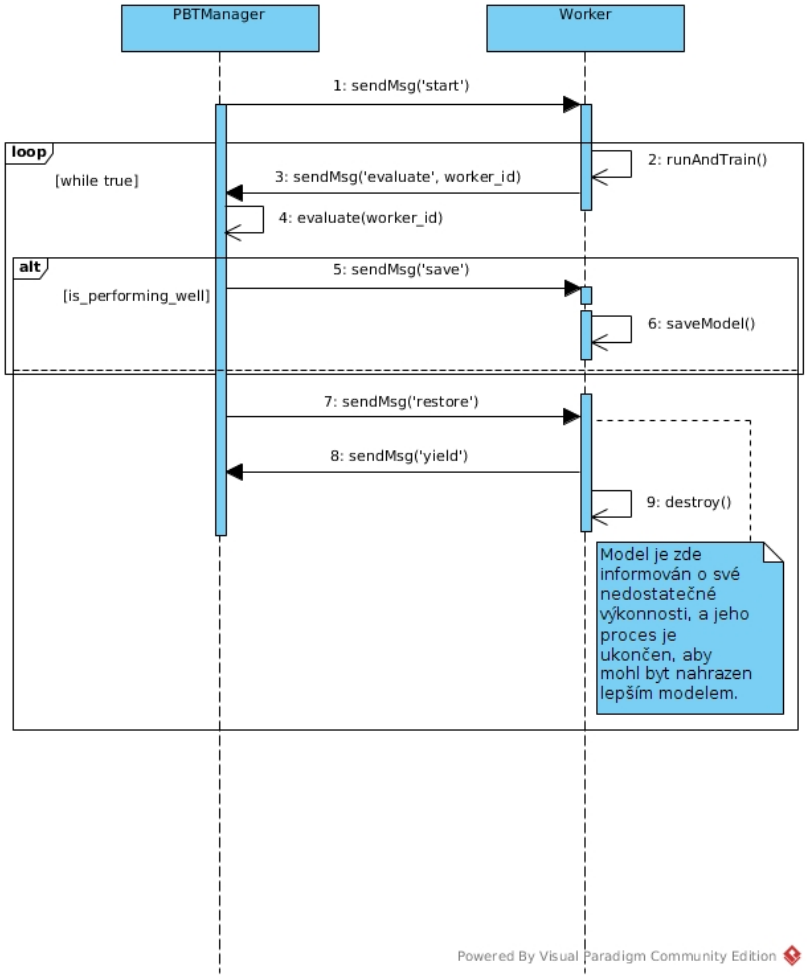
Jednotlivé modely manažerovi periodicky ohlašují svou úspěšnost, která je ukládána a průběžně aktualizována, přičemž je také dále vyhodnocována oproti ostatním modelům. Na základě této evaluace je prostřednictvím nakonfigurované a zvolené metodiky rozhodnuto, zda bude daný model uložen ve svém současném stavu[1] nebo přepsán více úspěšným modelem a jakým.

Při přepisování modelu je již na manažerovi, aby se postaral o korektní nahrazení modelu. Po vyhodnocení, že daný model nesplňuje požadavky na úspěšnost, volí manažer rovnou i náhradní model. Poté předává nevyhovujícímu modelu informaci o tom, že má být ukončen a nahrazen úspěšnějším modelem[2]. Poté je u nahrazovaného modelu nejdříve nutné, aby se zbavil zodpovědnosti za správu přidělených prostředí, proto je před opětovným sestavováním modelu předán odkaz na přidělenou skupinu prostředí PBT manažerovi. Model pak informuje hlavní proces o ukončení své činnosti, uvolňuje přidělenou paměť a ukončuje svůj proces.

V této fázi již může hlavní proces vytvořit nový model, který přenáší váhovou distribuci neuronové sítě toho nahrazujícího, přičemž získává identifikační údaje a skupinu simulačních

[1] Ukládání se provádí pomocí check-pointů.

[2] Pro nahrazení je nutné ukončit proces modelu a odstartovat nový, kvůli uvolnění dedikované paměti na grafické kartě. Knihovna tensorflow-gpu neumožňuje provést pouhé nahrazení modelu. U CPU verze knihovny tensorflow tento postup není nutný, ale pro znovupoužitelnost je zachován, jelikož nepředstavuje tak závažný výkonnostní bottleneck, aby stálo za uvážení ponechat pouze cpu verzi.



Obrázek 7.2: Sekvenční diagram zachycuje komunikaci mezi entitou `PBTManager` a `Worker`, tedy proces vytváření modelu, jeho evaluace, ukládání a případného nahrazování.

prostředí toho předchozího. Model náhodně inicializuje hyperparametry, uloží check-point svého stavu, a pokračuje v trénování nanovo. Tento postup vyúsťuje v efektivní prohledávání nadějné části váhového prostoru s rozdílnou dynamikou způsobenou změnou v hodnotách hyperparametrů (pro detailnější popis viz 3.2).

7.3.2 Třída Worker

Dostáváme se k třídě `Worker`, která představuje kontaktní bod mezi PBT manažerem, agentem jednoho konkrétního modelu a správou simulace přiřazených prostředí. Po inicializaci konfiguračních parametrů a obdržení přidělených prostředí je zapotřebí sestavit dva klíčové aktéry: `ActorCriticAgent` a `Runner`.

Třída agenta slouží v první řadě ke správě samotného modelu. `ActorCriticAgent` vytváří umělou neuronovou síť na základě předaných parametrů specifikujících kromě hyperparametrů například použitou strategii, optimalizační algoritmus, atd. Síť je vytvořena s pomocí knihovny `tensorflow` a veškeré interakce s ní probíhají prostřednictvím `tensorflow.Session`. Zároveň zprostředkovává i práci s vestavěným systémem pro ukládání a nahrávání modelů `tensorflow.Saver`.

Hlavním úkolem třídy `Runner` je pak zajistit snadnou komunikaci mezi agentem a prostředími SC2. Zajišťuje korektní provedení fáze vnímání (získání informací o aktuálním stavu z prostředí a předání těchto informací agentovi v požadované formě) a fáze akce (předání této akce danému prostředí). `Runner` vždy provádí akce nad prostředím s využitím agenta po batchích, akumuluje vypozorované návratové hodnoty, a parsuje je pro následovné učení.

Po sestavení obou těchto klíčových částí algoritmu může `Worker` odstartovat běh hlavní smyčky (viz výpis kódu 7.1). Kromě zajištění hladkého běhu simulace prostředí a procesu trénování se tento proces zabývá i pravidelným logováním záznamů o vývoji učení a zejména výměnou informací s PBT manažerem při evaluaci výkonu samotného modelu. Na základě zpětné vazby od tohoto hlavního procesu se dále rozhoduje, zda má `Worker` uložit svůj stav a pokračovat v trénování, nebo se zbavit svých dedikovaných prostředí a ukončit svůj proces. Hlavní smyčka tedy vypadá následovně:

```
while True:
    if i % 1000 == 0:
        Worker._print(i)
    training_input = self.runner.run_batch(self.envs, self.agent) # run
    if flags.training:
        self.agent.train(training_input) # train
    else:
        pass
    i += 1
    if i % self.batches_per_eval == 0:
        done = self.evaluate_and_update_model(i)
    if 0 <= flags.episodes <= self.runner.episode_counter or done:
        break
```

Výpis 7.1: Hlavní smyčka běhu a učení konkrétního modelu

7.3.3 Třída SC2Env

`SC2Env` poskytuje možnost jednoduše vytvořit a pracovat s prostředím prostřednictvím přímočarého rozhraní pro jazyk python. Dává k dispozici četná nastavení vytvářené hry, ať už se jedná o:

- Výběr mapy,
- nastavení obtížnosti,
- systém skóre,
- volbu možné vizualizace průběhu hry,
- atd.

Prakticky veškerý kód obsažený v této třídě je převzat z původní implementace (není naprosto žádný důvod jej měnit). Vše obsažené v této třídě a ve třídách, které používá, má zpřístupnit hru SC2 pro programovatelný subjekt vstupující do ní jako hráč. Není tedy nutné se jí zde detailněji zabývat.

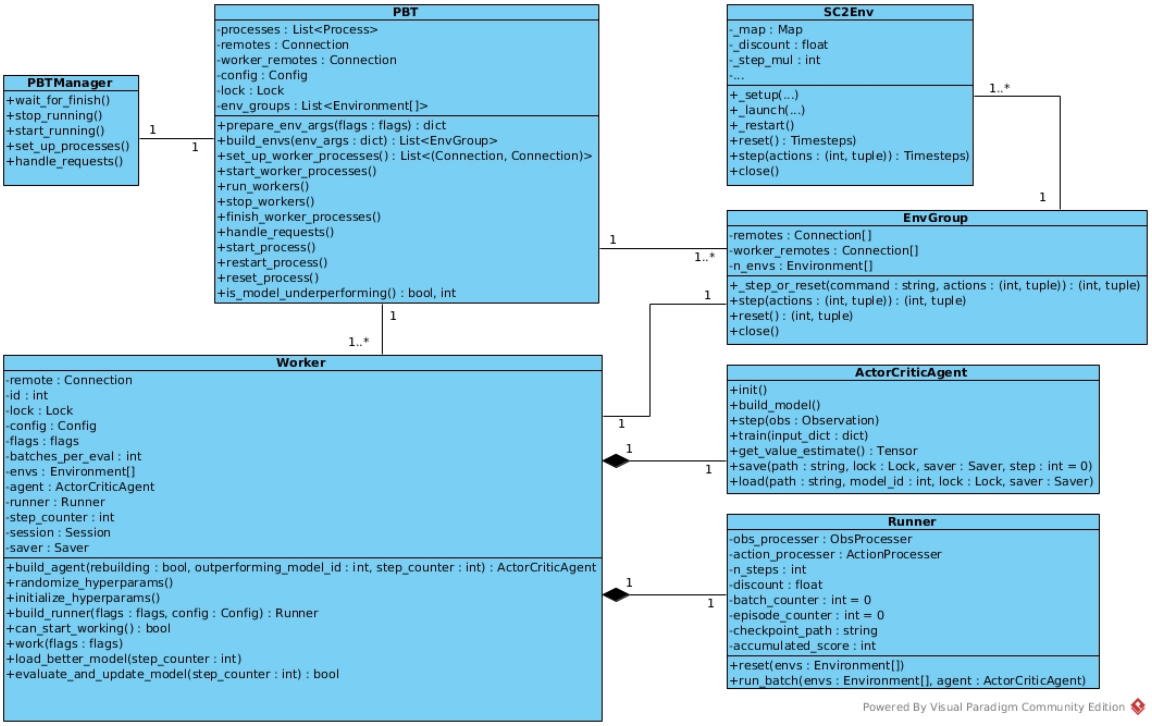
7.3.4 Ucelený pohled

Nyní, když jsme si vysvětlili funkcionalitu jednotlivých částí, by bylo záhodno nastínit detailnější pohled na architekturu. `PBTManager`, `Worker` i `SC2Env` mají své vlastní procesy, a komunikují mezi sebou zasíláním zpráv s pomocí pythonovské knihovny `multithreading`. Diagram tříd na obrázku 7.3 popisuje vztahy mezi jednotlivými důležitými částmi programu. Nejdůležitějším use-case je jednoznačně proces trénování, proto sekvenční diagram na obrázku 7.4 ukazuje do hloubky, jak jednotliví aktéři zasahují do trénovacího procesu.

7.4 Trénování na CPU oproti GPU

V průběhu implementace vyvstala na povrch řada problémů, které byly nejčastěji způsobeny buď vysokou úrovní paralelizace programu, nebo z důvodu velmi vysoké náročnosti algoritmu jak z pohledu času, tak požadavků na výkon HW. Experimenty s dostatečně velkou populací trvají v nejlepším případě pár dní, a proto bylo velmi důležité odbourat chyby co nejdříve. Jako příklad implementačního problému zde uvádím rozhodování mezi procesorem a grafickou kartou pro trénování neuronové sítě.

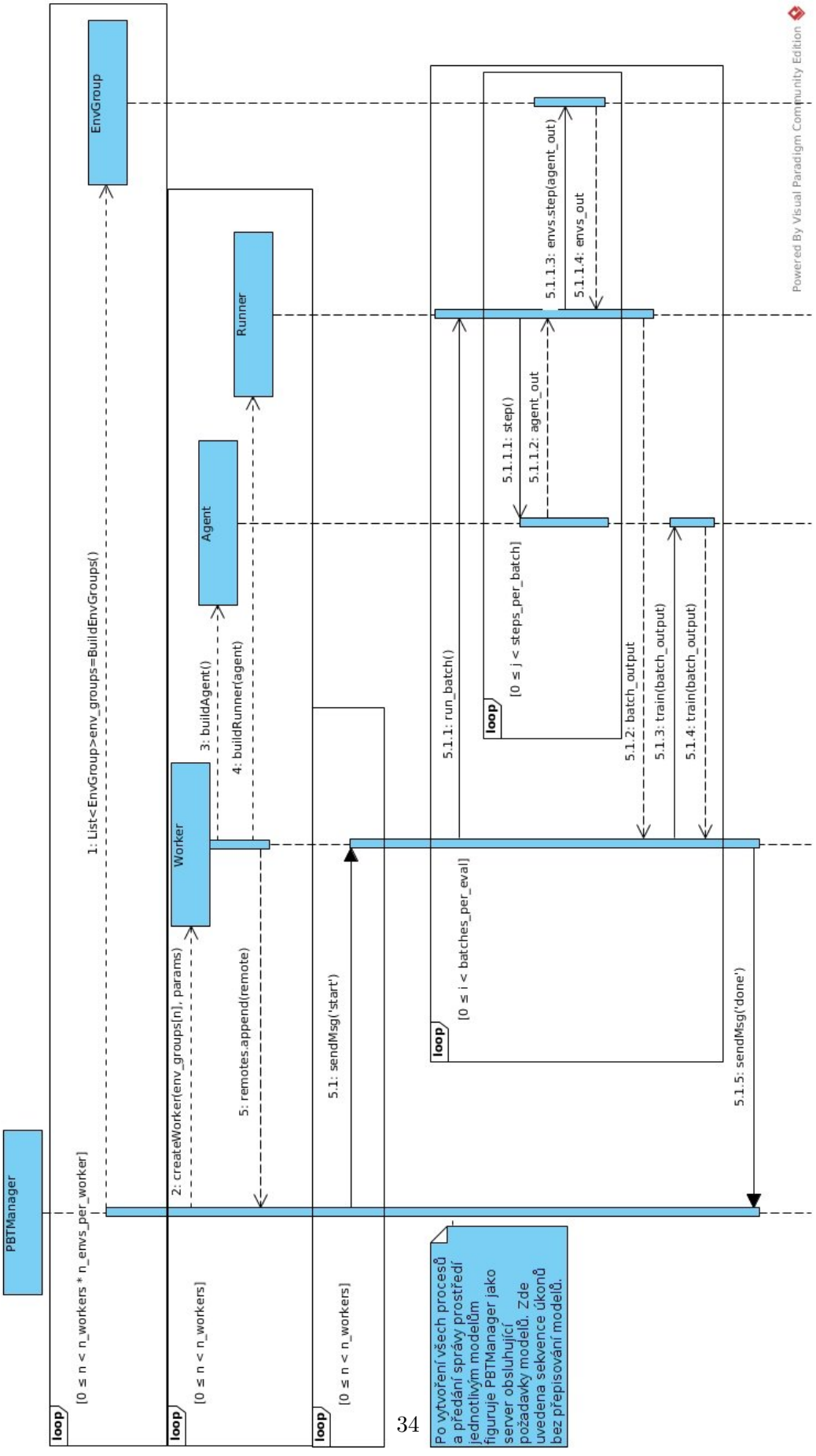
Prostředí zaměřující se na provádění výpočtů na grafické kartě vyžaduje větší režii a vyhrazuje si velké množství paměti (jak té na grafické kartě, tak té operační). Oproti tomu ale zaznamenává několikanásobně vyšší rychlost trénování (z pokusů bylo vypozorováno pětinásobné až dvanáctinásobné zrychlení podle použité konfigurace, a to zejména co se týče množství simulovaných modelů a výběru herního scénáře). Toto zrychlení vychází samozřejmě z převedení velké části výpočtů z procesoru na grafickou kartu. Požadavky týkající se paměti jsou však natolik vysoké, že i velmi silné grafické karty mají velký problém simulovat více než 4 modely najednou (experimentálně ověřeno na grafické kartě NVIDIA GeForce GTX 1080 Ti s pamětí 11GB GDDR5X), což z metody trénování na GPU dělá metodu nepoužitelnou (Population Based Training vyžaduje paralelní trénování alespoň 20 modelů [20]). Postup založený na GPU s sebou nesl kromě toho i další problémy týkající se správy paměti na grafické kartě, přepisování modelů podle PBT, apod. Stále však v průběhu



Obrázek 7.3: Diagram tříd. Za zmínku hlavně stojí, že třídu `Worker` zastřešující konkrétní proces spojený s jedním modelem tvoří třída `ActorCriticAgent` představující agenta a třída `Runner`, která zprostředkovává rozhraní s přiřazenou skupinou prostředí `EnvGroup`.

implementace hrál díky své rychlosti důležitou roli při verifikaci správnosti algoritmu např. při ověřování funkčnosti interakčních mechanismů mezi vlastním programem a prostředím Starcraftu II nebo procesu trénování pomocí A2C. Zároveň však našel své využití i při testování funkcionality PBT implementace s nízkou populací, jelikož byl již při návrhu kladen velký důraz na škálovatelnost algoritmu.

Oproti tomu řešení založené na využití CPU jako hlavní výpočetní jednotky sice pracuje mnohem pomaleji, nevykazuje však stejné nedostatky týkající se paměti, zejména tedy nepotřebuje ke své činnosti paměť GPU, ale i požadavky na operační paměť jsou nižší. Pro experimenty tedy využívám implementace založené čistě na CPU i přes výkonnostní nedostatky oproti GPU verzi, jelikož je schopna trénovat i několikrát více modelů než pouze 4.



Obrázek 7.4: Detailní sekvenční diagram zachycující proces trénování bez zahrnutí nahrazování modelů skrze PBT.

Kapitola 8

Experimentální fáze a diskuze

Nejdříve se v následující podkapitole zaměříme na specifikaci HW prostředí, ve kterém bylo testování a experimentování prováděno. Pak bude následovat vymezení konfigurace trénovaných modelů a architektury umělé neuronové sítě, kterou se snažíme učit. V postupném sledu budou následovat jednotlivé experimenty.

8.1 HW prostředí

Jak již bylo uvedeno v podkapitole 7.4, pro většinu experimentů práce využívá CPU implementace algoritmu, a tedy z pohledu rychlosti trénování hrají klíčovou roli zejména dva faktory:

- Výkon procesoru - jasný požadavek. Řešení představované v této práci staví na vysoké paralelizaci, a tedy lze plně využít vícejádrové procesory.
- Velikost operační paměti RAM - rychlý přístup ke všem potřebným datům je naprosto klíčový pro efektivní trénování. Nutnost přistupování k disku a opakované přepisování bloků v RAM tvoří kritický bottleneck, a podle množství nedostačující paměti se může trénování zpomalit i řádově (ověřeno i experimentálně).

Na provedení většiny testů určených pro verifikaci implementace vystačil vlastní hardware (Intel Core i5-6300HQ 2.30GHz, 8GB RAM paměti, grafická karta NVIDIA GeForce GTX 950M). Zároveň byl pro otestování možností a požadavků GPU implementace využit fakultní stroj FIT VUT s názvem *pcfajcik*. Jeho parametry (Intel Core i5-6500 3.20GHz, 8GB RAM, NVIDIA GeForce GTX 1080 Ti) byly to nejlepší, co bylo v rámci fakulty možné využít pro trénování na grafické kartě. Zároveň sloužil jako alternativní stroj k ověření přenositelnosti algoritmu.

Podíváme-li se na CPU implementaci, bylo mi umožněno využít fakultního stroje *athena6*, jelikož má spolu se stejnojmennými stroji na FIT VUT nejvyšší výpočetní sílu (šestijádrový procesor Intel Xeon E5-2630 2.60GHz a přibližně 128GB RAM). Přesto bohužel výkon tohoto stroje nepostačoval při provádění experimentů pro plné využití technologie PBT (plně vytížený stroj zvládal ve všech experimentech maximálně 20-25 modelů při nízkém počtu simulovaných prostředí na jeden model, doporučuje se zvolit až 40 modelů pro kvalitní výsledky, 20 modelů je stanoveno jako minimum pro získání použitelných výsledků [20]). V tomto ohledu tedy konečné výsledky experimentů nemusejí odpovídat potenciálně kvalitnějším a stále dosažitelným výsledkům.

8.2 Konfigurace

Architektura umělé neuronové sítě v jádru agenta odpovídá FullyConv síti popsané v podkapitole 6.1.2. Vytvoření a ověření této architektury není předmětem práce, a tedy se jí zde nebudu blíže zabývat. Použití stejné síťové architektury umožňuje přesnější porovnání s modelem vytvořeným společností DeepMind.

Nezávisle na tom však existuje velké množství konfigurovatelných parametrů hrajících důležitou roli v procesu trénování. Lze je rozdělit do typů podle oblasti využití, a v této podkapitole si uvedeme detailně jejich vliv na běh programu a trénovací proces.

8.2.1 Populační parametry

Populační parametry ovlivňují svým způsobem velikost populace, konkrétně se jedná o počet trénovaných modelů a počet prostředí jim přiřazených. Jak již bylo zmíněno dříve, PBT vyžaduje alespoň 20 modelů k vykazování pozitivního vlivu populačního přístupu k optimalizaci hyperparametrů. Počet prostředí přidělených jednotlivým modelům se vztahuje vždy na jeden model, a tedy každý model má pro simulaci k dispozici stejný počet prostředí. V průběhu experimentování se osvědčilo zejména zvyšovat počet prostředí u GPU implementace, jelikož jejich simulace probíhá pouze s využitím CPU (a samozřejmě RAM paměti), což znamenalo minimální nárůst časové náročnosti. Pro tuto verzi tedy i více než 10 simulovaných prostředí na model nepředstavovalo problém z hlediska nároků na výpočetní sílu. Naopak CPU implementace zřejmě zaznamenávala propady ve výkonnosti při větším počtu prostředí na model, a pro 20 a více modelů znamenalo jakékoliv navýšení přídělu výrazný pokles rychlosti trénování. Za optimální počet (vzhledem k dostupnému HW) bylo stanoveno 4 prostředí na každý z modelů pro jakýkoliv prováděný seriózní experiment s vysokou populací.

Tyto parametry zjevně zdaleka nejvíce ovlivňují výpočetní náročnost. Kvůli důležitosti obou parametrů budou vždy u každého experimentu uvedeny jejich hodnoty. V této chvíli je také velmi důležité poznamenat, že i provedení takto omezených experimentů trvalo vždy řádově minimálně pár dní (pro získání dostatečně kvalitních výsledků), a maximálně klidně i více než 10 dní za plného vytížení stroje.

8.2.2 Parametry prostředí

Volba konkrétního scénáře (tedy mini-hry, případně některé z map pro plnou hru SC2) představuje velmi důležitý konfigurační prvek hned z několika důvodů. Zaprvé, konkrétní scénář ovlivňuje strukturu odměňování, zřejmě si totiž každá z mini-her klade za cíl ověřit schopnosti učení a úspěšnosti sítě v alespoň mírně pozměněné problémové doméně. Mini-hry mohou změnu stavu odměňovat pozitivně i negativně (kupříkladu `DefeatRoaches` přisuzuje body hráči za každou zničenou jednotku *Roach*, oproti tomu v průběhu hry trestá jakoukoliv ztrátu vlastní jednotky *Marine*). Scénáře také dále vymezují množinu proveditelných akcí a množství vypozorovatelných jevů ze stavu hry, jelikož např. přímo omezují typy vyskytujících se jednotek v daném scénáři, nezakomponovávají do mapy žádný terénní profil, či přímo zakazují použití některých akcí.

API také umožňuje vizualizaci simulované hry, při trénování by to však bylo kontraproduktivní[1]. Nastavení ostatních parametrů spadajících do této kategorie opisuje jejich

[1] Jistě je jasné, že vykreslování desítek simulovaných prostředí není smysluplné využití prostředků při trénování sítě.

vymezení v původní práci ([2]), jelikož pouze upravuje interakci s prostředím, a pro porovnání s předchozími pracemi by bylo záhodno tyto parametry neměnit. Jedná se kupříkladu o velikost intervalu počtu herních kroků (snímků) před provedením každé akce agentem (viz 5.2) nebo rozlišení pozorovaných feature vrstev aktuálního stavu. Pokud bude s těmito nastaveními jakkoliv manipulováno s ohledem na standardní hodnoty, uvádím tuto skutečnost explicitně u konkrétního experimentu.

8.2.3 Síťové parametry

Pro tuto práci představují síťové parametry klíčovou položku. Kromě jejich primární funkce nasměrování sítě k úspěšnému natrénování se tyto parametry stávají klíčovou položkou pro diferenciaci modelů v průběhu trénování na základě průběžného hodnocení úspěšnosti. Zaměřuje se na ně i ukládání a přepisování modelů v rámci metody PBT, protože je v nich zakódován i postup dosažení úspěšnosti daného modelu. Specifickou roli mají zde váhové parametry pro počáteční inicializaci (použito uniformní rozložení). Jsou stanoveny a použity pouze při počátku trénovacího procesu, jelikož PBT se stará při nahrazování modelů právě o přenášení vah mezi úspěšnými a neúspěšnými modely. Oproti tomu ostatní hyperparametry, jako je learning rate, discount, max norm gradient nebo entropie pro rozložení pravděpodobnosti provádění akcí, mohou svou hodnotu měnit při nahrazování modelů. Stanovení jejich výchozích hodnot pramení jak z původní práce, tak z vlastních experimentů. Konfigurační soubor specifikuje hranice uniformní distribuce možných inicializačních hodnot pro každý z uvedených hyperparametrů. Rozdíly v těchto hodnotách pak tvoří klíčový faktor při kvalitě a rychlosti trénování, avšak je třeba vzít v potaz, že optimální hodnoty těchto parametrů se mohou v průběhu trénovacího procesu měnit, a PBT se právě tento fakt snaží řešit. Při každém přepisování modelu tedy horší model vždy převezme váhový profil úspěšnější sítě, zatímco ostatní hyperparametry inicializuje nezávisle na svém lépe natrénovaném vzoru.

Pro většinu experimentů jsou položeny následující hodnoty:

- Learning rate - 10^{-7} až 10^{-3}, vždy s postupnou degradací hodnoty (prostřednictvím zvoleného optimalizačního algoritmu *Adam*).
- Discount faktor - 0.95 až 0.999 s uniformním rozložením.
- Max norm gradient - 150 až 700 s uniformním rozložením.
- Podíl entropie na volbě konkrétní akce a souřadnic pro prostorové akce - obojí 10^{-6} až 10^{-2} s uniformním rozložením.

Opět platí stanovisko, že pokud byl v průběhu některý z těchto parametrů měněn, bude to uvedeno v popisu experimentu.

8.2.4 PBT parametry

Při správě populace modelů je důležité stanovit si nějakým způsobem pravidla, podle kterých se bude vyhodnocovat jejich úspěšnost, a potažmo podmínky určující, jak zvolit nahrazující a nahrazovaný model. Přesně tímto problémem se zabývají PBT parametry. Jedním z nich je stanovení počtu provedených batchů (jejichž velikost se taktéž uvádí), které musí být zpracovány před zažádáním o vyhodnocení daného modelu oproti ostatním. Tento hyperparametr přímo ovlivňuje frekvenci vyhodnocování, ukládání a přepisování modelů,

a v průběhu experimentování s ním bylo často manipulováno, jako nejúspěšnější se však osvědčilo stanovit tuto hodnotu na 10000 batchů před každou evaluací.

Ve vlastní režii pak byla určena kritéria pro vyhodnocování modelů. Podle referencí a vlastního experimentování se jako dobrá volba jevilo zakomponování dvou faktorů: práh vztažený k pořadí modelů na základě úspěšnosti, a rozdíl mezi právě vyhodnocovaným a nejlepším modelem populace, opět ve vztahu k jejich individuální úspěšnosti v průběhu posledního nepřetržitého trénovacího intervalu. Nejlepších výsledků bylo dosaženo při nahrazování nejhorších 20% modelů náhodným výběrem z 10% těch nejlepších při splnění podmínky, že mezi nahrazovaným a nahrazujícím modelem byl rozdíl v kumulativní úspěšnosti alespoň 10%. V opačném případě se jednoduše místo nahrazení model ukládá a pokračuje se v trénování.

8.2.5 Dávkovací parametry

Do této kategorie spadají údaje týkající se např. velikostí batchů, frekvence logování výsledků, ale hlavně zde patří počet vyhodnocovaných epizod před ukončením trénování. Tento poslední zmiňovaný údaj se samozřejmě mění a bude buď vždy u konkrétního experimentu uveden nebo je jeho hodnota stanovena na 25000 epizod.

8.3 Testování úspěšnosti implementace

Jako nejsmysluplnější způsob testování kvality implementace se jeví porovnání úspěšnosti v mini-hrách, které mají jasná pravidla a strukturu odměňování, a výsledky jsou snadno vyčíslitelné a porovnatelné. Síť bude natrénována k hraní jednotlivých mini-her, a výsledky budou porovnány s referenčními řešeními Deep Mind. Doposud neexistuje žádný RL model, který by dokázal vyvinout smysluplnou strategii pro hraní plné hry Starcraft II, a to ani proti vestavěným skriptovaným botům s nastavením nejnižší obtížností. Vítězství RL agenta v SC2 by bylo obrovským úspěchem v této komplexní doméně i proti tomuto soupeři.

8.4 Experiment srovnání různých konfigurací

Jak lze vidět v kapitole 8.2, existuje velké množství parametrů, které různým způsobem ovlivňují prakticky veškeré aspekty trénování. Vzhledem k náročnosti algoritmu však je nutné brát v potaz i rychlost získání uspokojivého výsledku a nastavení konfigurace s touto rychlostí velmi úzce souvisí. Uveďme si zde jako příklad výběr mezi CPU a GPU implementací [2]. Experiment byl proveden v prostředí mini-hry `MoveToBeacon`, přičemž algoritmus byl spuštěn v třech konfiguracích, které se od sebe liší následovně:

- 1. konfigurace - 4 modely po 4 prostředích trénované na GPU.
- 2. konfigurace - 4 modely po 4 prostředích trénované na CPU.
- 3. konfigurace - 20 modelů po 4 prostředích trénované na CPU

Každá z konfigurací byla spuštěna třikrát, a jejich výsledky byly zprůměrovány. Zajímá nás zejména náročnost z pohledu času a počtu epizod potřebných pro dosažení uspokojivého výsledku.

[2] Bez použití PBT, jelikož GPU nedokáže efektivně simulovat běh většího množství modelů, viz podkapitola 7.4.

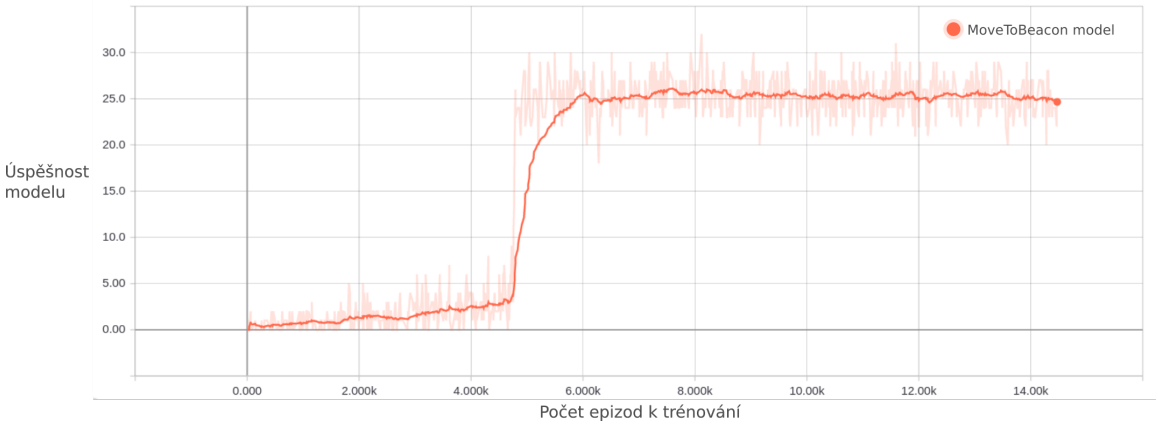
	4m4e - GPU	4m4e - CPU	20m4e - CPU
Délka trénování	27m	2h36m	06h38m
Počet epizod	3120	3717	1246

Tabulka 8.1: Srovnání náročnosti trénování konfigurací lišících se v použitém HW a počtu trénovaných modelů. Ukončovací podmínkou je stanoveno dosažení hodnoty 25 jako průměrné úspěšnosti v rámci jednoho trénovacího cyklu, a porovnává se, kdy jí daná populace dosáhne u alespoň jednoho ze svých modelů.

Pointou zvolené mini-hry je navigovat jedinou jednotku na velmi omezeném prostoru směrem k tzv. beaconu. Jakmile jej jednotka dosáhne, hráč je odměněn jedním bodem, a beacon se přemístí do jiného místa na mapě. Za cíl mini-hry se považuje co největší možný počet navštívení beaconu před vypršením časového limitu. Tento scénář omezuje akční prostor pouze na jedinou prostorovou akci přesunu vybrané jednotky, přičemž správná volba prostorové parametrizace akce se stává hlavním úkolem trénovacího procesu. Zároveň ale přesunování beaconu probíhá náhodně (se započtením určité minimální vzdálenosti nově vytvořeného beaconu od aktuální pozice hráče). Toto představuje i pro RL agenta poměrně triviální záležitost, a `MoveToBeacon` zůstává doposud jedinou mini-hrou, ve které vítězí umělá inteligence nad lidským protějškem.

Z výsledků testování vyplývá již zmíněná užitečnost využití grafické karty při trénovacím procesu, čímž se efektivně odděluje trénování prováděné na GPU od veškeré režie a simulace prostředí, které má na starosti CPU. Experiment ale ukázal nedeterminismus při trénovacím procesu, což lze vyčíst z rozdílu hodnot mezi počtem epizod GPU a CPU verze algoritmu s čtyřmi trénovanými modely (rozdíly jsou ještě markantnější při porovnání individuálních pokusů, jelikož uvedené hodnoty jsou zprůměrované).

Zároveň si lze všimnout, že i pro takto triviální úkol dosažení vysoké průměrné úspěšnosti stanovené na hodnotu 25 může trvat velmi dlouho. Algoritmus je však navržen tak, aby maximálně využíval možností paralelizace, a při korektním využití více procesorů se tento čas rapidně zkrátí.

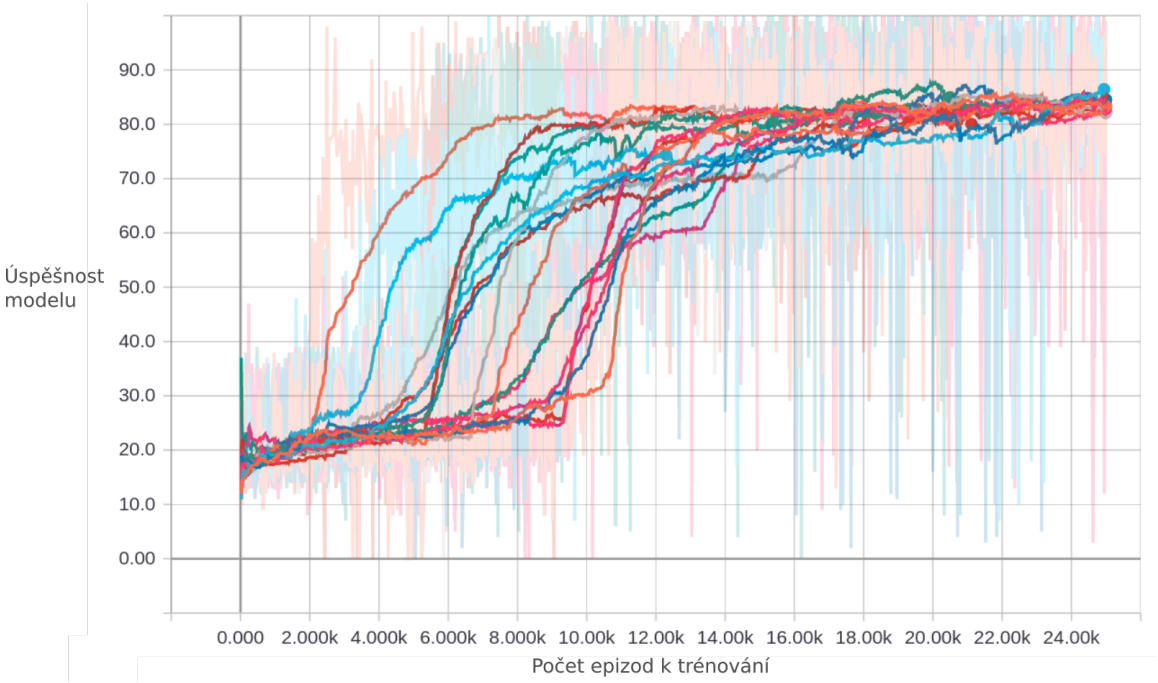


Obrázek 8.1: Typický příklad průběhu trénování jednoho konkrétního modelu pro mini-hru `MoveToBeacon`. Různé trénovací procesy i při stejně zvolených hyperparametrech dosahují optimální úspěšnosti v různém čase, což je přímý důsledek nedeterminismu učení z dat vytvářených za běhu v simulovaných prostředích.

8.5 Experiment vyhodnocení výhod použití PBT

Tato práce si neklade za úkol zeširoka ověřovat validitu využité metody, přestože se jedná o velmi čerstvě publikovanou záležitost (prosinec 2017). I tak by ale bylo záhodno ověřit přidanou hodnotu použití PBT v prostředí SC2, a porovnat ji s některou z dříve používaných metod paralelního hledání.

Velkou výhodou je možnost měnění konfigurace i v průběhu trénování. Při učení umělé neuronové sítě můžou nastat nežádoucí situace, která může být vyřešena přenastavením hyperparametrů. Jako příklad si zde můžeme uvést případy, kdy trénovací proces uvázne v lokálním maximu, nebo přestává konvergovat k lepšímu řešení. Jak už bylo zmíněno v podkapitole 3.2, PBT se snaží tento problém řešit nahrazením neúspěšných modelů úspěšnými, a tím nejen odbourává zbytečně investovaný výpočetní čas, ale zároveň může nahrazený model s odlišnými hyperparametry prohledávat stejný váhový prostor svého kopírovaného vzoru a předčit jej. V průběhu učení se totiž optimální hodnoty hyperparametrů pro efektivní trénování mohou měnit.

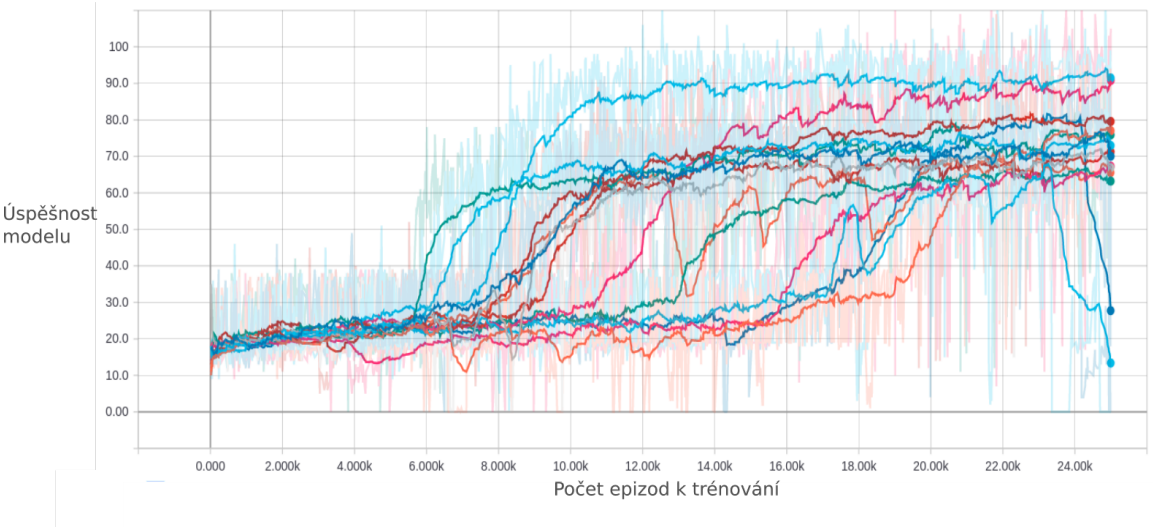


Obrázek 8.2: Příklad průběhu trénování populace 20 modelů s optimalizací hyperparametrů pomocí PBT.

Je ovšem nutné poznamenat, že to samo o sobě nezaručuje, že takováto manipulace s hyperparametry za běhu zaručí zlepšení celkové úspěšnosti. Podívejme se například na graf z obrázku 8.2. Lze vidět, že pokud určitý model vykazuje rapidní zvýšení produktivity, zatímco ostatní zůstávají nepříliš úspěšné, tak se postupnou evaluací v průběhu trénování začínají nahrazovat neúspěšné modely úspěšnějším. V tomto běhu se však stalo, že žádný z modelů dále nepokračoval ve zvyšování produktivity i nadále[3]. Oproti tomu se můžeme podívat na vybraný běh bez využití PBT pro optimalizaci hyperparametrů, který je ilu-

[3]Samozřejmě se může stát (a je to i pravděpodobné, soudíme-li z ostatních běhů), že pokud by trénování pokračovalo i po nakonfigurovaných 25000 epizodách, některý z modelů najde cestu k vyšší úspěšnosti.

strovaný na obrázku 8.3. Hned na první pohled lze vypozorovat, že dva modely v tomto případě předčily populaci modelů z výše zmíněného snímku. Oproti němu však k relativně správnému řešení došlo mnohem méně modelů, a PBT populace tedy má vyšší pravděpodobnost k nalezení optimálního řešení, přesto však nezaručuje, že toto lepší řešení najde (což platí zejména pro nízké populace).



Obrázek 8.3: Příklad průběhu trénování populace 20 modelů bez využití PBT.

Z výsledků a uvedených grafů lze tedy vyvodit, že PBT rozhodně (alespoň pro menší populace) nezajišťuje rychlejší dosažení lepších výsledků než paralelní učení, protože se stále jedná o proces, který do značné míry investuje při rozhodování do entropie. I pokud ale vezmeme tento fakt v potaz, v konečném důsledku PBT nabízí flexibilnější přístup k učení, které vyústí v obecně kvalitnější výsledky, což lze vidět v tabulce 8.2. Proto se tento přístup jeví jako optimálnější, a vzhledem k nízké režii v porovnání s nutným procesem trénování má určitě své využití.

Jednou z klíčových výhod použití tohoto systému pro optimalizaci hyperparametrů je také mnohem rychlejší hledání užitečných konfigurací. U takto složitých problémů může tento proces s využitím obyčejného pararelního hledání obvykle trvat i několik týdnů až měsíců. PBT průběžně identifikuje nevyhovující konfigurace a přesune jejich pozornost a dedikovanou výpočetní sílu do perspektivnějšího prostoru, přičemž zároveň dokáže identifikovat i případy, kdy se vyplácí změnit hyperparametry až v pozdější fázi učení. Kromě referenčních řešení tedy i tento faktor výrazně pomohl při hledání spolehlivějšího konfiguračního schéma pro jednotlivé mini-hry.

8.6 Porovnání úspěšnosti modelů s DeepMind FullyConv agentem

Podívejme se nyní, jaké úspěšnosti dosáhla vlastní implementace oproti DeepMind FullyConv modelu a referenčnímu A2C agentovi. Jsou uvedeny údaje ze všech mini-her, které bylo možné ve vymezeném čase vyhodnotit a natrénovat na nich vlastní řešení. Všechny uve-

	Vlastní impl.		Ref. agent A2C		DM FullyConv	
	Průměr	Max.	Průměr	Max.	Průměr	Max.
MoveToBeacon	26	33	25	N/A	26	35
CollectMineralShards	96	125	93	N/A	103	134
DefeatRoaches	89	355	87	N/A	100	355
FindAndDefeatZerglings	43	51	42	N/A	45	56

Tabulka 8.2: V tabulce jsou uvedeny údaje o úspěšnosti vlastního řešení k porovnání s referenčním A2C agentem [32] a A3C FullyConv agentem společnosti DeepMind [2]. Za průměrné skóre se považuje průměr dosažený modelem s nejlépe zvolenými hyperparametry, maximum pak indikuje nejlepší dosažený výsledek.

dené výsledky experimentů týkající se vlastního agenta jsou získány spuštěním s následující konfigurací:

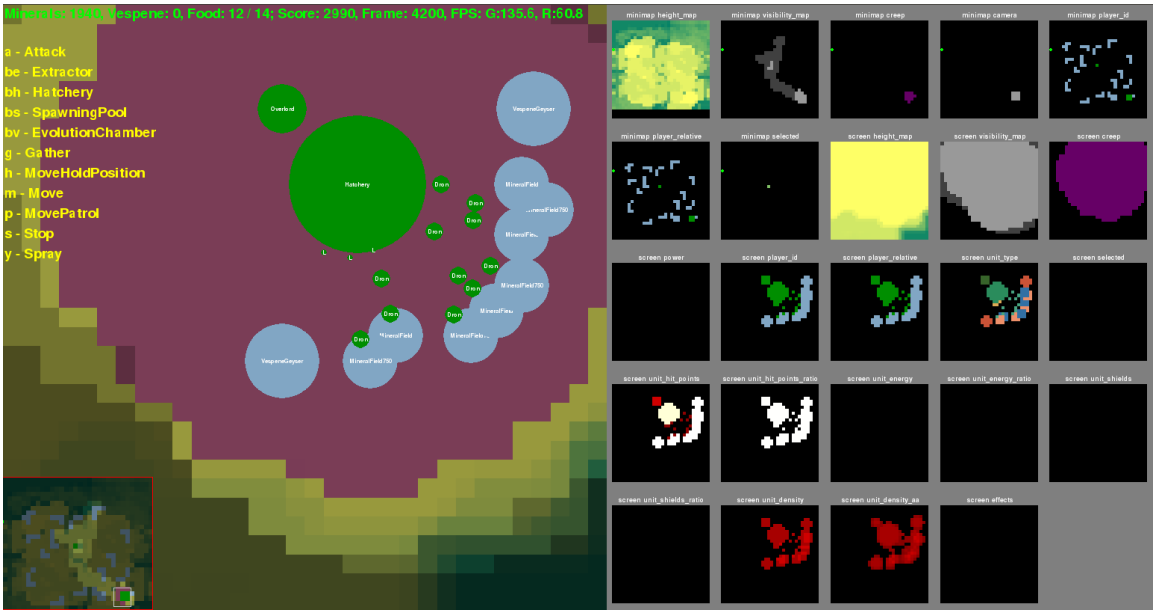
- Celkem 20 trénovaných modelů v populaci.
- Každému modelu přiřazena 4 prostředí pro simulaci běhu hry.
- Jednotlivé běhy mají nastaven limit 25000 epizod pro trénování.
- Ostatní parametry nastaveny na výchozí hodnoty uvedené v podkapitole 8.2.

Z výsledků můžeme vypozorovat, že ač se vlastní implementace svou úspěšností blíží agentovi společnosti DeepMind, tak kromě mini-hry `MoveToBeacon`, která je koncepčně dostatečně jednoduchá k nalezení optimálního řešení, náš agent nedosahuje stejných výsledků. Oproti tomu v porovnání s referenčním agentem A2C prakticky nezávisle na scénáři lze vypozorovat zlepšení v řádu jednotek procent. To přisuzuji zejména mechanismu prohledávání hyperparametrického prostoru pomocí PBT.

8.7 Plná hra

Jak již bylo popsáno, plná hra SC2 představuje prozatím nedosažitelný cíl i pro state-of-the-art agenty využívajících zpětnovazebného učení. Agenti nejsou schopni odvodit logický postup pro vítězství ve hře i proti slabým oponentům, a dosavadní řešení neumějí nabídnout žádnou užitečnou strategii. Přesto se nabízí provést experiment pro otestování tohoto tvrzení použitím vlastní implementace. Vzhledem k odlišnému přístupu práce s hyperparametry může síť hledat řešení i v prostoru, ve kterém se FullyConv nepohyboval. Byla sestavena populace o 20 modelech po 4 simulačních prostředích, s limitem 12000 epizod před ukončením trénování. Jako scénář pro simulaci byla zvolena mapa pro dva hráče `AbyssalReef`, která se běžně používá při hráčských kláních na oficiálním žebříčku pro hru jednoho proti jednomu. Ostatní parametry byly ponechány podle podkapitoly 8.2. Struktura odměňování byla založena na nativním skóre vestavěným do hry SC2, na základě kterého síť manipulovala při trénování s váhovými parametry.

I přes nastavení méně než poloviního limitu pro počet vykonávaných epizod bylo trénování velmi náročné a dlouhé, a po 10 dnech běhu bylo manuálně ukončeno kvůli nedostatku času. Podle očekávání nedokázal ani jeden z modelů uspět v plné hře proti jednomu hráči vestavěné umělé inteligence nejnižší obtížnosti. Po ukončení trénování byly jednotlivé modely blíže zkoumány skrze vizualizační rozhraní PySC2 prostředí. Z tohoto zkoumání vyplynulo,



Obrázek 8.4: Screenshot z průběhu hry na mapě `AbyssalReef`.

že ty nejúspěšnější modely (v porovnání s ostatními) dokázaly pouze kumulovat nativní skóre prostřednictvím stoprocentního soustředění na těžbu minerálů, což garantovalo stálý přísun bodů. Jediná variace v chování s pozitivním efektem byla dedikování malého množství jednotek k průzkumu mapy. Umělá neuronová síť tak preferovala objevení nejpřímočařejšího přístupu k stabilnímu přísunu skóre bez ohledu na vítězství ve hře. Takový přístup samozřejmě v žádném případě nelze považovat za správný, a z obecného hlediska jde o uváznutí v lokálním extrému. Blíže budou výsledky tohoto experimentu komentovány v diskuzi.

Je také nutné poznamenat, že vzhledem k neschopnosti algoritmu dosahovat uspokojivých algoritmů v plné hře nedává smysl zúčastnit se konkrétní soutěže v jakémkoliv turnaji mezi boty SC2. Zároveň je ale nutno poznamenat, že komunita Starcraft AI se stále koncentruje okolo starší verze Starcraftu, a SC2 turnaje prakticky nejsou k vidění.

8.8 Diskuze

Abychom plně pochopili uvedené výsledky, musíme si ujasnit vliv faktorů vstupujících do procesu trénování toho nejlepšího modelu, který lze pomocí vlastní implementace využívající zmiňovaných metodik vytvořit. V první řadě je nutné uvést, že experimentální fáze nebyla dostatečně důkladná k tomu, aby neexistoval prostor pro nalezení lepšího řešení i pouhým prodloužením trvání trénovacího procesu. To je způsobeno zejména nedostatkem času pro experimentování, což vyplývá z vysoké výpočetní náročnosti běhu algoritmu. Tomuto problému by bylo možné se vyvarovat rychlejší analýzou hardwarových požadavků, které ovšem bylo nutné získat také experimentálně, jelikož většina technologií spojených s touto prací byla publikována velmi nedávno a navržená implementace vykazuje velmi proměnlivé požadavky závislé na aktuální konfiguraci. Experimentální fáze tak byla prováděna na nejlepším dostupném fakultním hardware (avšak stále nedostatečně výkonném), a to jak v případě CPU verze, tak i při testování možností GPU verze. Jako důsledek se prakticky veškeré trénovací běhy pohybovaly délkou trvání minimálně v řádu dnů, což se

promítalo negativně na nutnosti omezení délky experimentálních běhů na snesitelnou hranici, na době zjišťování a opravování chyb vyskytujících se pouze při určitých konfiguracích běhů a v neposlední řadě na omezení velikosti populace na snesitelnou úroveň. Všechny tyto faktory mají přímý vliv na finální úspěšnost vykazovanou algoritmem. Uvedené výsledky dosažené i přes vypsané nedostatky tedy nemusí odpovídat nejlepším možným výsledkům, kterých lze dosáhnout. Prakticky provedený benchmarking patří nyní také k výstupům této práce, a pro další budoucí výzkum je možné využít poznatků zde.

Pomineme-li výše uvedené a zaměříme-li se na opravdu dosažené výsledky, tak můžeme vypozorovat navýšení úspěšnosti napříč mini-hrami oproti referenčnímu A2C agentovi o několik procent, a v porovnání s FullyConv agentem společnosti DeepMind stále vlastní implementace mírně zaostává ve většině případů. Agent byl však schopen ve všech čtyřech případech nalézt validní strategii, a úspěšně plnit cíle stanovené daným scénářem, což se projevilo i při následovném pozorování chování algoritmu s využitím vizualizačního rozhraní PySC2.

8.8.1 Plná hra a další směr vývoje

Může existovat několik důvodů, proč síť dochází k výsledkům uvedeným v podkapitole 8.7, ze zkušeností s touto problematikou a z dat vycházejících z experimentů lze odvodit následující možné problémy:

- Nesprávný systém odměn pro zpětnovazebné učení - Nativní skóre použité jako odměna nemá dostatečně spolehlivou vypovídací hodnotu týkající se aktuálního stavu hráče v porovnání s oponentem. Zároveň může dojít velmi často k výskytům situací, kdy odměna za konkrétní akci buď přichází až s odstupem, nebo akce způsobuje periodický přísun odměn (jako se to děje například u těžby). V takových případech může agent nesprávně vyhodnotit akci jako užitečnou a takové chování má pak negativní dopad na proces učení. Tento faktor má s největší pravděpodobností nejvyšší vliv na neschopnost sítě naučit se komplexnějším úkonům, které mohou vyústit k útočení na protivníka.

- Nepřiměřená komplexita problému pro použitou síť - I oproti složitějším mini-hrám se plná hra jeví jako nesrovnatelně komplexnější problém jak z pohledu zahrnutí více informací z vypozorovaného stavu pro rozhodovací proces, tak z pohledu velikosti a rozmanitosti samotné množiny akcí, ze kterých si síť může vybírat směr dalšího postupu.

- Nesprávné rozložení inicializačních intervalů pro hyperparametry sítě - Přestože PBT dokáže najít kvalitní inicializační hodnoty pro hyperparametry, stále je nutná specifikace intervalů, ve kterých by se tyto hodnoty měly pohybovat. Vzhledem k diametrálnímu rozdílu v komplexitě mezi mini-hrami a plnou hrou může být tato specifikace validní pro hodnoty značně odlišné, než tomu bylo u mini-her.

Podívejme se na první dva z uvedených problémů blíže, včetně návrhu řešení obou problémů. Ze shrnutí týkajícího se systému odměn výše vyplývá, že agent upřednostňuje nejjednodušší možný přístup k stabilnímu získávání bodů, bez ohledu na vítězství. Problém nastává ve chvíli, kdy užitečný úkon vyžaduje příliš dlouhou sekvenci akcí. Síť v této chvíli buď nedokáže sekvenci správně vykonat dostatečně často ve správném kontextu (aby vedla k pozitivním výsledkům), nebo nedokáže spolehlivě přiřadit získanou odměnu k této sekvenci kvůli příliš zpožděnému odměňování. Klíčová myšlenka, která shrnuje neschopnost

sítě vykazovat sofistikovanější strategii, je podle autora následující tvrzení: Zvolený systém odměn nereflektuje měnící se priority spojené s prováděním akcí v závislosti na fázi hry a konkrétní herní situaci. Síť pak kvůli tomu neumí vydedukovat proces postupného zlepšování směřujícímu k provádění strategicky užitečných akcí v dané situaci. Konkrétně existuje příliš velká bariéra mezi nežádoucím setrváním v pasivním hraní soustředěném na sběr surovin a žádoucím přechodem na inkluzi interakcí s protivníkem v rozhodovacím procesu.

Volba správného systému odměn je podle autora klíčovým směrem dalšího vývoje, který může přinést lepší výsledky. Alternativní systém by mohl například eliminovat periodické odměňování vyplývající z těžby, a pouze přisuzovat odměnu při počátku těžby (a naopak ji může odebírat při ukončení těžby). To ulehčí síti identifikovat užitečné akce s větší přesností. Další praktickou vlastností může být postupné měnění struktury odměňování v průběhu hry. Těžba představuje kritickou činnost v začátcích, postupem času se ale více pozornosti v ideálním případě přesouvá směrem k tvorbě útočných a obranných jednotek, a tedy by se s postupem času mělo odměňování zaměřit spíše na vytváření vlastních jednotek a budov, a ničení těch nepřátelských.

Můžeme také tvrdit, že vybraná síťová struktura nemá kapacitu na to, aby se dokázala vypořádat s obrovskou komplexitou simulace hraní plné hry. V takovém případě by se situace dala řešit dekompozicí problému. Kupříkladu několik sítí může zvlášť dirigovat jednotlivé aspekty hry, přičemž by existoval rozhodovací mechanismus (může se opět jednat o umělou neuronovou síť), který by zajišťoval výběr sítě pro provedení následující akce. Stejně tak může teoreticky mít každá ze sítí vlastní strukturu odměňování v závislosti na své specializaci. Hraní plné hry SC2 zahrnuje řešení podproblémů v průběhu hry (průzkum, rozšíření základen, vylepšování jednotek,...) stejně tak jako balancování při rozdělování pozornosti mezi jednotlivými podproblémy, taková hierarchie tedy logicky dává větší smysl.

Kapitola 9

Závěr

Práce nejdříve usiluje o zhodnocení stavu a využití strojového učení, a to jak z historického hlediska, tak z pohledu state-of-the-art technologií. Přibližuje současná propojení umělé inteligence s různými odvětvími, a zároveň přibližuje roli strategických her v pokroku současného vývoje v oblasti strojového učení.

Hlavním bodem zájmu pro strojové učení ve strategických hrách je aktuálně Starcraft II, a na základě toho byl navržen a vytvořen agent zpětnovazebného učení, který kombinuje rámec pro zpětnovazebné učení A2C a optimalizační algoritmus PBT k dosažení co nejlepších výsledků v této doméně. Bylo experimentálně ověřeno, že vlastní řešení představuje konkurenceschopného agenta pro referenční řešení, přestože mírně zaostává za doposud nejúspěšnějším state-of-the-art publikovaným agentem vydaným společností Google DeepMind. Vlastní implementace dosahovala kvalitních výsledků v mini-hrách, zatímco v plné hře (stejně jako všechna ostatní řešení založená na zpětnovazebném učení) neuměla odvodit validní strategii k porážce svého protivníka. Na základě experimentů byl navržen další postup, včetně stanovení možných směrů budoucího vývoje. Zpětnovazebné učení nadále představuje princip s největším potenciálem v tomto odvětví. Stále ale existuje mnoho prostoru pro vylepšování stávajících algoritmů i způsobu přístupu k této problémové doméně.

Literatura

[1] *OpenAI baselines github repository*. [Online; navštíveno 09.01.2018].
URL https://github.com/openai/baselines

[2] *PySC2 github repository*. [Online; navštíveno 09.01.2018].
URL https://github.com/deepmind/pysc2

[3] *SC2API github repository*. [Online; navštíveno 09.01.2018].
URL https://github.com/Blizzard/s2client-proto

[4] Abadi, M.; Barham, P.; Chen, J.; aj.: TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, s. 265–283, [Online; navštíveno 20.01.2018].
URL
https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[5] Baxt, W. G.: Use of an Artificial Neural Network for Data Analysis in Clinical Decision-Making: The Diagnosis of Acute Coronary Occlusion. MIT Press - Journals, prosinec 1990, s. 480–489.
URL https://doi.org/10.1162/neco.1990.2.4.480

[6] Beattie, C.; Leibo, J. Z.; Teplyashin, D.; aj.: DeepMind Lab. *CoRR*, 2016.
URL http://arxiv.org/abs/1612.03801

[7] Bellemare, M. G.; Naddaf, Y.; Veness, J.; aj.: The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, ročník 47, 2013, doi:10.1613/jair.3912.
URL http://doi.org/10.1613/jair.3912

[8] Bernstein, A.; de V. Roberts, M.: Computer v. Chess-Player. *Scientific American*, ročník 198, č. 6, 1958: s. 96–107, ISSN 00368733, 19467087.

[9] Brown, N.; Sandholm, T.: Libratus: The Superhuman AI for No-limit Poker. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, AAAI Press, 2017, ISBN 978-0-9992411-0-3, s. 5226–5228.
URL http://dl.acm.org/citation.cfm?id=3171837.3172067

[10] Campbell, M.; Hoane, A.; hsiung Hsu, F.: Deep Blue. *Artificial Intelligence*, ročník 134, č. 1-2, leden 2002: s. 57–83, doi:10.1016/s0004-3702(01)00129-1.

[11] Copeland, J.: *Artificial Intelligence: A Philosophical Introduction*. Wiley-Blackwell, 1993, ISBN 063118385X.

[12] Crevier, D.: AI: The Tumultuous History Of The Search For Artificial Intelligence. Basic Books, 1993, ISBN 0465029973, str. 96.

[13] Crevier, D.: AI: The Tumultuous History Of The Search For Artificial Intelligence. Basic Books, 1993, ISBN 0465029973, str. 109.

[14] Dormehl, L.: *A history of artificial intelligence in 10 landmarks*. 2017, [Online; navštíveno 05.01.2018].
URL https://www.digitaltrends.com/cool-tech/history-of-ai-milestones/

[15] French, R. M.: Subcognition and the Limits of the TuringTest. *Mind*, ročník XCIX, č. 393, 1990: s. 53–65, doi:10.1093/mind/xcix.393.53.
URL https://doi.org/10.1093/mind/xcix.393.53

[16] Goertzel, B.; Pennachin, C.: *Artificial General Intelligence (Cognitive Technologies)*. Springer, 2007, ISBN 354023733X, 6–8 s.

[17] Huang, S.: Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG). leden 2018.
URL
https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

[18] Hyunwoo, O.; Teguh, B.; Yi, D.; aj.: Identifying the Rush Strategies in the Game Logs of the Real-Time Strategy Game StarCraft-II. *Training*, 2017: str. 306.

[19] J. Berliner, H.: Chess Report - Deep Thought Wins Fredkin Intermediate Prize. ročník 10, leden 1989: s. 89–90.

[20] Jaderberg, M.; Dalibard, V.; Osindero, S.; aj.: Population Based Training of Neural Networks. *CoRR*, ročník abs/1711.09846, 2017, 1711.09846.
URL http://arxiv.org/abs/1711.09846

[21] Kaelbling, L. P.; Littman, M. L.; Cassandra, A. R.: Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, ročník 101, č. 1-2, květen 1998: s. 99–134, doi:10.1016/s0004-3702(98)00023-x.
URL https://doi.org/10.1016/s0004-3702(98)00023-x

[22] Lewis, T.: *A Brief History of Artificial Intelligence*. 2014, [Online; navštíveno 05.01.2018].
URL https:
//www.livescience.com/49007-history-of-artificial-intelligence.html

[23] Lou, H.; McArdel, S.: *AI in Video Games: Toward a More Intelligent Game*. 2017, [Online; navštíveno 05.01.2018].
URL http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/

[24] Mauldin, M. L.: ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, American Association for Artificial Intelligence, 1994, ISBN 0-262-61102-3, s. 16–21.
URL http://dl.acm.org/citation.cfm?id=199288.199285

[25] McCorduck, P.: *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. A K Peters/CRC Press, 2004, ISBN 1568812051.

[26] Mnih, V.; Badia, A. P.; Mirza, M.; aj.: Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, ročník abs/1602.01783, 2016, 1602.01783.
URL http://arxiv.org/abs/1602.01783

[27] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: Human-level control through deep reinforcement learning. *Nature*, ročník 518, č. 7540, únor 2015: s. 529–533, doi:10.1038/nature14236.
URL https://doi.org/10.1038/nature14236

[28] Moor, J. H.: An analysis of the turing test. *Philosophical Studies*, ročník 30, č. 4, říjen 1976: s. 249–257, doi:10.1007/bf00372497.

[29] Moravcík, M.; Schmid, M.; Burch, N.; aj.: DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *CoRR*, 2017.
URL http://arxiv.org/abs/1701.01724

[30] Morris, P.: Enhance The Power And Functionality Of Siri With SiriToggles For iPhone 4S. leden 2012.
URL http://www.redmondpie.com/enhance-the-power-and-functionality-of-siri-with-siritoggles-for-iphone-4s-download-now/

[31] Newell, A.; Shaw, J. C.; Simon, H. A.: *Chess-Playing Programs and the Problem of Complexity*. Springer New York, 1958, ISBN 978-1-4613-8716-9, s. 89–115, doi:10.1007/978-1-4613-8716-9_7.

[32] Pekka Aalto: *pekaalto github repository*. [Online; navštíveno 09.01.2018].
URL https://github.com/pekaalto/sc2aibot

[33] Peng, P.; Yuan, Q.; Wen, Y.; aj.: Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games. *CoRR*, 2017, [Online; navštíveno 20.01.2018].
URL http://arxiv.org/abs/1703.10069

[34] Poole, D. L.; Mackworth, A. K.: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2017, ISBN 110719539X.

[35] Ramesh, A.; Kambhampati, C.; Monson, J.; aj.: Artificial intelligence in medicine. Royal College of Surgeons of England, září 2004, s. 334–338, doi:10.1308/147870804290.

[36] Samuel, A. L.: Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.*, ročník 3, č. 3, 1959: s. 210–229, ISSN 0018-8646, doi:10.1147/rd.33.0210.

[37] Saygin, A. P.; Cicekli, I.; Akman, V.: *Minds and Machines*, ročník 10, č. 4, 2000: s. 463–518, doi:10.1023/a:1011288000451.

[38] Schaeffer, J.; Lake, R.; Lu, P.; aj.: Chinook: The World Man-Machine Checkers Champion. ročník 17, únor 1996.

[39] Shannon, C. E.: XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, ročník 41, č. 314, březen 1950: s. 256–275, doi:10.1080/14786445008521796.

[40] Silver, D.; Schrittwieser, J.; Simonyan, K.; aj.: Mastering the game of Go without human knowledge. *Nature*, ročník 550, č. 7676, říjen 2017: s. 354–359, doi:10.1038/nature24270.

[41] Synnaeve, G.; Bessière, P.: A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. *CoRR*, 2011, [Online; navštíveno 16.01.2018].
URL http://arxiv.org/abs/1111.3735

[42] Tesauro, G.; Sejnowski, T.: A Parallel Network that Learns to Play Backgammon. ročník 39, červenec 1989: s. 357–390.

[43] Tromp, J.; Farnebäck, G.: Combinatorics of Go. In *Computers and Games*, Springer Berlin Heidelberg, 2007, s. 84–99, doi:10.1007/978-3-540-75538-8_8.

[44] Vinyals, O.; Ewalds, T.; Bartunov, S.; aj.: StarCraft II: A New Challenge for Reinforcement Learning. *CoRR*, 2017.
URL http://arxiv.org/abs/1708.04782

[45] Vlček, M.: *Predikce deště z meteoradaru*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2015.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=15828

[46] Wu, H.; Zhang, J.; Huang, K.: MSC: A Dataset for Macro-Management in StarCraft II. *CoRR*, 2017, [Online; navštíveno 17.01.2018].
URL http://arxiv.org/abs/1710.03131

[47] Wu, L.; Markham, A.: Evolutionary Machine Learning for RTS Game StarCraft. *AAAI*, 2017, [Online; navštíveno 18.01.2018].

[48] Wu, Y.; Mansimov, E.; Liao, S.; aj.: OpenAI Baselines: ACKTR & A2C. Srpen 2017, [Online; navštíveno 16.01.2018].
URL https://blog.openai.com/baselines-acktr-a2c/

[49] Zobrist, A. L.: A Model of Visual Organization for the Game of GO. In *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference*, AFIPS '69 (Spring), ACM, 1969, s. 103–112, doi:10.1145/1476793.1476819.

Příloha A

Obsah přiloženého paměťového média

- `run_sc2.py` - hlavní skript pro spouštění běhu programu.
- `config.py` - konfigurační soubor.
- `core` - adresář se zdrojovými soubory.
- `_files` - adresář pro ukládání trénovaných modelů. V podsložce `models` jsou uloženy tři natrénované modely pro vyzkoušení.
- `LICENSE` - licenční ujednání.
- `README.txt` - manuál.
- `doc` - adresář obsahující text práce a plakát.

Příloha B

Manuál

Prerekvizity k použití CPU verze programu:

- `python 3.5`
- `tensorflow 1.5.0+` (pref. 1.7.0)
- `OpenAI baselines` (co nejaktuálnější)
- `Starcraft II` (implementováno a testováno na linux ubuntu 16.04) + mapy včetně minimap
- `tensorflow-tensorboard` pro přehledné zobrazení výsledků

Pro GPU verzi je prerekvizit více, všechny souvisí s instalací knihovny `tensorflow-gpu`.

Návod k použití:

```
python3 run_sc2.py
```

Veškeré možnosti konfigurace spuštění lze nastavit v souboru *config.py*. Zejména věnujte pozornost parametrům:

- `n_models` - počet modelů k trénování
- `n_envs` - počet prostředí Starcraft II přiřazených každému modelu k simulaci
- `if_output_exists` - nastavení chování v případě, že ve výstupní složce již existuje nějaký model
- `episodes` - délka trénování v epizodách na jeden model

Trénované modely se ukládají do složky `_files/models`. Výsledky trénování jsou k nalezení ve složce `_files/summaries`. Více informací v *config.py*.